

GROUSE: Generator of Research Output Units in Software Engineering

by

Abhijeet Mohanty

Bachelor of Technology, National Institute of Technology, Karnataka

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2021

Chicago, Illinois

Defense Committee:

Mark Grechanik, Chair and Advisor

Luis Gabriel Ganchinho de Pina

Cornelia Caragea

Copyright by
Abhijeet Mohanty
2021

ACKNOWLEDGMENT

First and foremost, I would like to express my gratitude towards my advisor, Professor Mark Grechanik for guiding me through the world of research, teaching me how to improve my problem solving capabilities and to always strive to be better than what I was.

Also, I would like to express my thanks to my committee members - Professor Pina and Professor Caragea for taking out time to serve on my defense committee.

Last but not the least, I owe it to my parents because of whom I could start my journey as a graduate student.

AM

TABLE OF CONTENTS

<u>CHAPTER</u>	<u>PAGE</u>
1 INTRODUCTION	1
2 MOTIVATION: CONCEPTUAL GRAPHS	6
2.1 Frame Based Models	7
2.2 Semantic Networks	7
2.3 Conceptual Graphs	7
3 RELATED WORK	14
4 TOOLS, NLP OPERATIONS AND TOPIC MODELLING TECHNIQUES	18
4.1 Natural Language Processing Operations	18
4.1.1 Tokenization	18
4.1.2 Part of Speech Tagging	18
4.1.3 Term Expansion	18
4.2 Topic Modelling	19
4.2.1 Latent Semantic Indexing	19
4.2.2 Singular Value Decomposition for LSA	19
4.2.3 Relational Topic Modelling	21
4.2.3.1 Parameters of RTM	22
4.2.3.2 Word and link generation process for documents	22
4.2.3.3 Computing the link probability function	23
4.2.3.4 Inference, estimation and prediction	23
4.2.4 Expected Entropy Loss	28
4.3 Important tools used	31
4.3.1 Natural Language Toolkit	31
4.3.2 WordNet	31
4.3.3 PdfBox	32
4.3.4 flask	32
4.3.5 marshmallow	32
4.3.6 scikit-learn	32
4.3.7 science-parse	33
4.3.8 pymongo	33
5 OUR CORPUS	34
5.1 Sections of a research paper	34
5.2 Authors	34

TABLE OF CONTENTS (Continued)

<u>CHAPTER</u>		<u>PAGE</u>
	5.3 Research questions	36
	5.4 References	36
	5.5 Content of the paper	38
6	SOLUTION	40
	6.1 Extracting textual content from papers	41
	6.2 Filtering terms in a paper	42
	6.3 Contextualizing terms in a paper	42
	6.4 Defining relations	42
	6.5 Building the conceptual graph	44
7	ARCHITECTURE	47
	7.1 Application design	47
	7.1.1 grouse-ui: The user interface	47
	7.1.2 grouse-db: The database	52
	7.1.2.1 grouse-db: Collections	59
	7.1.3 grouse-server: The backend	65
	7.1.3.1 application.py	65
	7.1.3.2 grouse/schema.py	66
	7.1.3.3 db/mongo_client.py	66
	7.1.3.4 sentence_parsing/utilities.py	67
	7.1.4 Using the application	67
	7.1.4.1 Deploying grouse-server	67
8	INFERENCE TECHNIQUES	73
	8.1 Building our collections	73
	8.2 Building the ground truth conceptual graph	73
	8.3 Concept graph creation through term expansion	73
	8.4 Concept graph creation using research questions	74
	8.5 Concept graph creation through Relational Topic Modelling	74
	8.6 Concept graph creation through Latent Semantic Analysis	75
	8.7 Concept graph creation through Expected Entropy Loss	75
9	CONCLUSION	76
	CITED LITERATURE	77
	VITA	83

LIST OF TABLES

<u>TABLE</u>		<u>PAGE</u>
I	Paper Sections' Metadata	35
II	Authors Metadata	35
III	Research Questions Metadata	36
IV	Most common nouns occurring in research questions	37
V	Citations Metadata	37
VI	Top 5 term frequencies	38
VII	Top 5 papers by no. of words	39
VIII	Miscellaneous paper metadata	39

LIST OF FIGURES

<u>FIGURE</u>		<u>PAGE</u>
1	A typical IS-A relationship	6
2	CG for <i>John is going to Boston by bus</i> (1)	8
3	CG for (2)	9
4	CG for (3)	10
5	CG for (4)	11
6	CG for (5)	12
7	CG for (6)	12
8	Semantic relationships (7)	43
9	Add term relations component	45
10	<i>GROUSE</i> flow diagram	47
11	Add category component	48
12	Add paper term category component	48
13	Add paper category component	49
14	Add relation category component	49
15	Merge categories component	50
16	View relations component	50
17	Build concept matrix component	51
18	Project, organization and language specification	53
19	Cloud service provider and cluster configuration	54
20	Enabling data access	55
21	Picking the connection method	56
22	Connect to the cluster	56
23	Create collections	57
24	Add own data	58
25	Enter database name and collection name	59
26	Parent directory of grouse-server	68
27	Create environment	68
28	Create application	68
29	Select environment tier	69
30	Filling in the application and environment name along with the domain	70
31	Uploading application code	71
32	Edit Network card	71
33	Configuring the VPC and the instance subnet	72

SUMMARY

Unlike mass media news articles, research papers have a well-defined structure and the language is much more formal. Many words have specific meanings and these meanings do not change based on the context in which these words are used (e.g., repository always designates a kind of storage in a source code control system).

We theorize that with a limited manual effort these research papers can be turned into a semantic graph where new relations between terms can be obtained. Moreover, with the vocabulary expansion, these terms can be linked to other concepts (e.g., repository \longrightarrow storage \longrightarrow memory) thereby enhancing the power of inferential reasoning about new relations. We also explore the possibility of finding contradictions among established relations hence raising new research questions about deeper explorations of already obtained relations.

Therefore, in this thesis, we address a problem of automatically creating inferences from the corpus of data published in research papers in the area of empirical software engineering.

To this end, we build an *MVC-based* application called **"Generator of Research Output Units in Software Engineering"** a.k.a. *GROUSE* which elicits the user to define relations between terms belonging to the corpus of 603 research papers published across various editions of Mining Software Repositories' conferences. With limited manual effort, we first create a ground truth conceptual graph upon which our application employs techniques such as *Expected Entropy Loss (EEL)*, *Latent Semantic Analysis (LSA)*, *Relational Topic Modelling (RTM)* and *term expansion* to infer new relations for semantically similar terms, terms

SUMMARY (Continued)

with similar underlying associations and papers which are related depending on the terms they contain.

With *GROUSE*, the empirical software engineering research community can collaborate on automatically generating new research questions that reveal deeper insights in software engineering processes and solutions.

CHAPTER 1

INTRODUCTION

Science is deeply fragmented, with many areas of specialization and researchers who work in these specialized areas present their works at designated venues, i.e., conferences and workshops and they publish in scientific journals. Reviewers, selected for these venues and journals, review submitted papers to determine if they satisfy quality standards for each specialized research area. Therefore, it is generally assumed that accepted papers contain novel research ideas and some proof of their viability. More generally, unlike publications in mass media or social media, these papers contain the terminology, the jargon and a pattern of discourse that are specific to each specialized research area. For example, when the word "architecture" is encountered in a software engineering paper, it is almost certainly meant as a formal blueprint for a software system and not for an urban building. Understanding the terminology and the jargon for each scientific area, and learning to use them in a proper discourse, is a part of education of every researcher.

Naturally, as part of learning the state-of-the-art in an area of science, researchers obtain a list of papers published at representative venues and analyze them. Once researchers understand the terminology and the structures of the papers, it becomes much easier for them to obtain "reusable brain artifacts," which are concise statements about the problem addressed in a given paper and their solutions. Even though there may be some disagreement about the meaning of

specific terms, it is rare enough to highlight that researchers mostly agree on big and important terms and relations among them.

While getting acquainted with the state of the art in a given area a researcher starts generating new ideas and hypotheses that can reveal new information to advance the state of the art in this area. One of the simplest argument to create new ideas and hypotheses is based on hypothetical syllogism, i.e., a researcher uses established facts, $P \longrightarrow Q$ from one paper and $Q \longrightarrow R$ from another paper to deduce a hypothesis that $P \longrightarrow R$. Often, the next step is to obtain the experimental data used in both papers to analyze it further to show that there is a statistically strong evidence to support the new hypotheses. As a result a new *Least Publishable Unit (LPU)* is born.

In this thesis we explore the possibility to infer new ideas and hypotheses from the corpus of peer-reviewed published research papers with a high degree of automation. We selected the research area of Empirical Software Engineering, specifically the subarea of mining software repositories. We collected over six hundred papers published over 12 years. We create a software application to assist the extraction of the Conceptual Graphs (CGs) from these papers to formalize domain-specific knowledge representation using the expert in Software Engineering, Prof. Mark Grechanik under whose guidance the work on this thesis was performed. These conceptual graphs along with the extracted technical terms are used as anchors to guide machine-learning (ML) algorithms for creating relationships among research questions raised in these papers, so that by exploiting these relationships we can suggest

The main idea of this project is to allow researchers to produce meta-research publications semi-automatically. Meta-research publications are based on existing publications that provided enough material to formulate research questions and use the previously published papers to answer these questions. As it is defined elsewhere, "Meta-analysis includes a set of methods that can combine quantitatively the evidence from different studies in a mathematically appropriate way. Combining data may improve statistical power, when there are several small studies on a specific question, but each one of them is largely under powered or has not been designed to address that research question. Meta-analysis may provide a precise and robust summary estimate after a systematic and rigorous integration of the available evidence." A different source provides a glimpse at the history of meta-analysis: "The American educational psychologist Gene V. Glass (1976) coined the term meta-analysis to stand for a method of statistically combining the results of multiple studies in order to arrive at a quantitative conclusion about a body of literature. The English statistician Karl Pearson (1857–1936) conducted what is believed to be one of the first statistical syntheses of results from a collection of studies when he gathered data from eleven studies on the effect of a vaccine against typhoid fever (1904). For each study, Pearson calculated a new statistic called the correlation coefficient. He then averaged the correlations across the studies and concluded that other vaccines were more effective than the new one. Our goal is to automate parts of the meta-analysis process to assist researchers in producing meta-research paper as least publishable units.

Consider an inference example for a new idea from several published papers. (2) can be described using a simple conceptual graph where repository is related to Java that is related to

library that is related to exception that is related to runtime that has bugs that are related to anti-patterns that are related to library and Java. The CG for (3) can be described as *Bugzilla* has bugs that have reports that have features and these reports that are related to NLP that has classification that is related to deontic norms that is related to behavior. (4) is characterized by the conceptual graph: *Jira* has issues that are related to developers that have sentiments and joy and love and *is-a* bullies that have productivity that *is-a* fixing time that *is-a* metric. (5) is characterized as a simple conceptual graph where repository is related to *Java* that is related to *library* or *application* that is related to exception that is `java.lang.Exception` that is related to handlers that has catch blocks and finally blocks that are related to patterns that are related to *errors* that *is-a* bug. Finally, (6) is about sentiment that has *SentiWordNet* that is related to *IT* and *ticket* that is related to developer that has feelings.

The word *deontic* is expanded with the synonyms ethics, morality, duty and obligations where morality and duty are strongly associated with feelings. From these conceptual graphs it is possible to infer the following hypotheses. First, exception has handlers that has catch and finally blocks and it is related to bugs that is related to anti-patterns and patterns. The latter is identified as contradiction for researchers to determine the relationship between using patterns vs anti-patterns on the number of bugs in exception handler blocks. Second, bug reports are related to morality and sentiment that is related to *IT* and *ticket* that is related to developer that is related to bugs that is related to exception handlers. Extending the first hypothesis a researcher can formulate a research question to determine if certain metrics of sentiments are prevalent in tickets and issues that are related to bugs in exception handlers. Since the

original papers provided empirical evaluations of the corresponding hypotheses, to answer the formulated research question requires a researcher to perform a number of standard steps to obtain the datasets and run analyses using software packages which are specified in the papers.

CHAPTER 2

MOTIVATION: CONCEPTUAL GRAPHS

To set the background of this chapter, it is necessary to understand the idea of conceptualism and its specifications. One such specification is that of an *ontology*. An ontology describes relations between various concepts governing some domain. These concepts in an object oriented parlance can be likened to classes, the attributes they contain and the relations they exhibit with other classes. An example is the following:

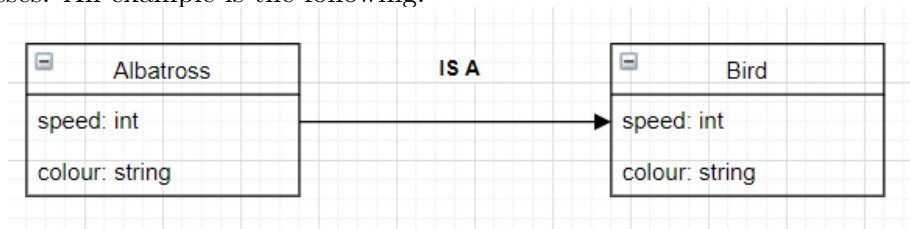


Figure 1: A typical IS-A relationship

In the above domain which concerns a *bird*, the concepts include its *attributes* - *colour* and *speed*, their *types* - *string* and *int* and the *relation* - *IS-A* between the two classes - *bird* and *albatross*.

For an intelligent system to understand ontologies, these ontologies ought to be represented formally. Formal representations typically define a relation between two concepts using which more implications can be drawn. There ought to be a certain taxonomy which governs the attributes a certain concept can possess and what other concepts can be derived by manipulating these attributes. A thing to keep in mind when choosing a formal representation is that it

should not be restrictive when representing concepts pertaining to the domain under interest.

We briefly describe frame based models, semantic networks and conceptual graphs.

2.1 Frame Based Models

In a frame based model, the knowledge representation primitive are frames and their properties. Here, frames denotes some primitive object that represents a concept in the domain under interest.

2.2 Semantic Networks

A semantic network is essentially a graph where each node denotes a concept and the edges represent the edges between these concepts. Such a network introduces the following relations:

1. *synonym* - The relation when two concepts A and B represent the same thing.
2. *antonym* - Here, the concept A represents something that is the opposite of what concept B represents.
3. *meronym*, *holonym* - Concepts through this relation express has-part and part-of relations.
4. *hyponym*, *hypernym* - Concepts can express the type-of relationship.

An important use case for this type of formalism is that of *interlingua* which is a representative language used to perform translations between different natural languages.

2.3 Conceptual Graphs

A conceptual graph is similar to a semantic network but it can be represented more directly as a first order predicate logic. The graphical representation of concepts further enables what the graph is trying to depict. As far as the graphical representation is concerned, rectangles

denotes some instance of a concept and an ellipse represents the relations between concepts. The directed edge shows in which direction the relation is oriented. An example of a conceptual graph is below:

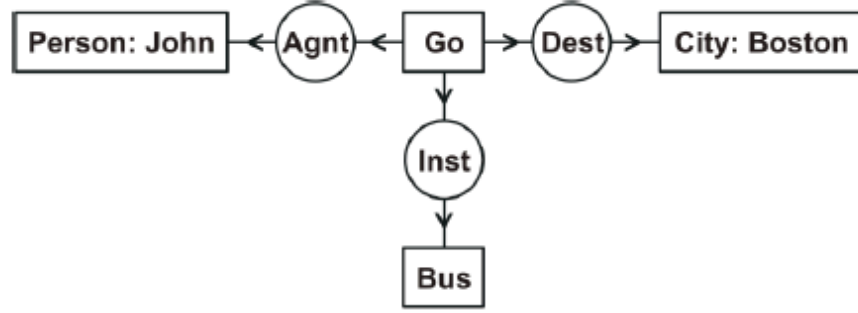


Figure 2: CG for *John is going to Boston by bus* (1)

Here, the concepts' labels are Person, Bus, City and Go and John and Boston denote concept names. The relations are *Agent* (*Agnt*), *Instrument* (*Inst*) and *Destination* (*Dest*) (1). The CG can be understood as a person being some agent of the concept Go which in turn uses the Bus as an instrument and destination as Boston. The first order predicate logic can be written as (1):

$$\begin{aligned}
 &(\exists x)(\exists y)(\text{Go}(x) \wedge \text{Person}(\text{John}) \wedge \text{City}(\text{Boston}) \wedge \text{Bus}(y) \wedge \\
 &\quad \text{Agnt}(x, \text{John}) \wedge \text{Dest}(x, \text{Boston}) \wedge \text{Inst}(x, y))
 \end{aligned}
 \tag{2.1}$$

Owing to the above logic, conceptual graphs can be understood by systems due to this formal representation. Going back to example in section 1, we can construct conceptual graphs as follows:

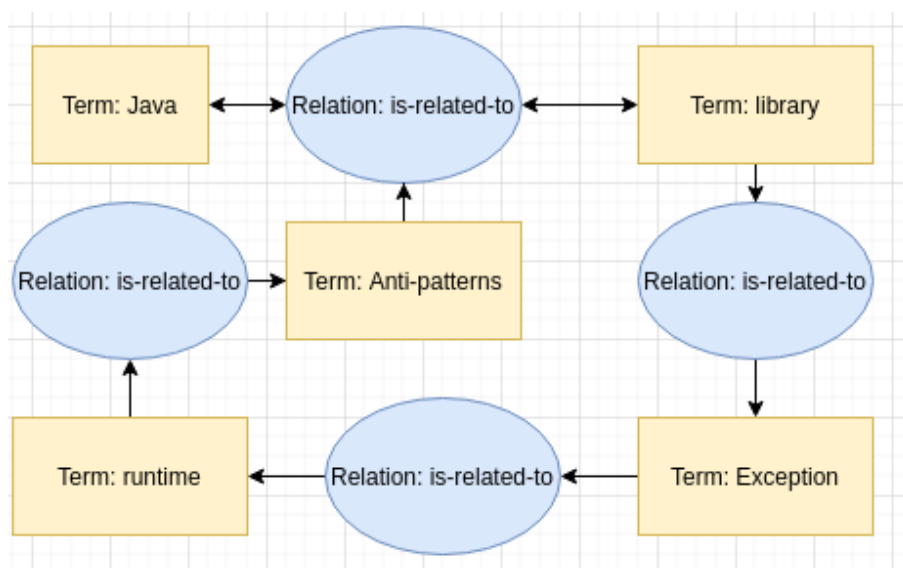


Figure 3: CG for (2)

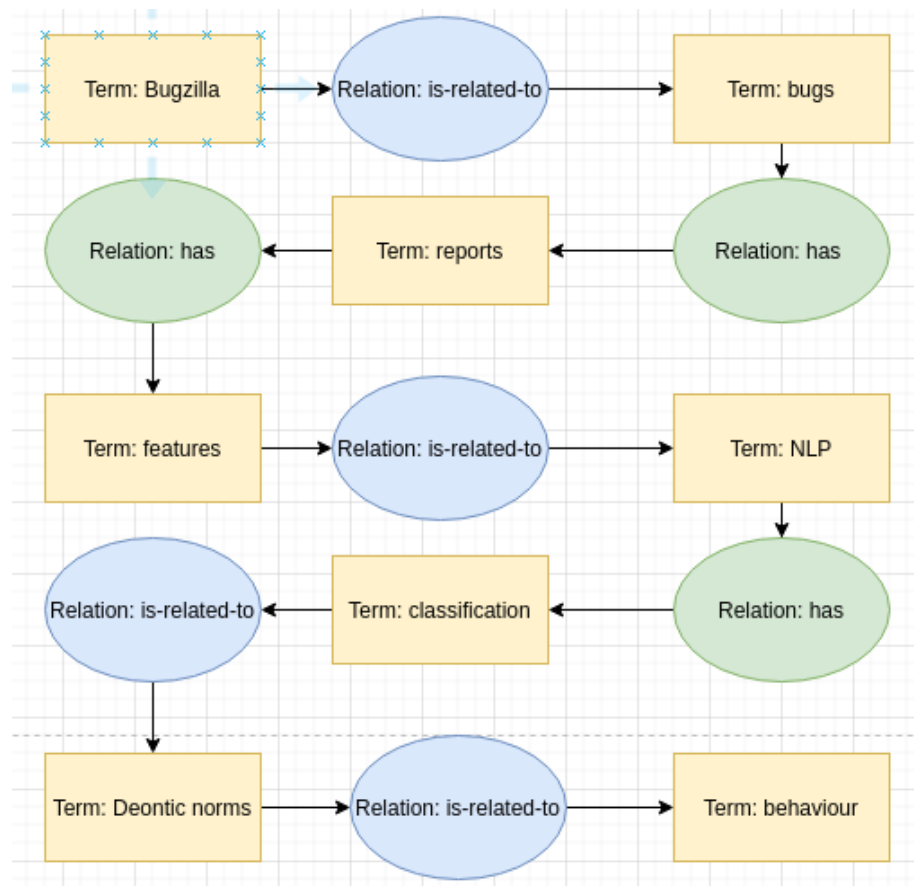


Figure 4: CG for (3)

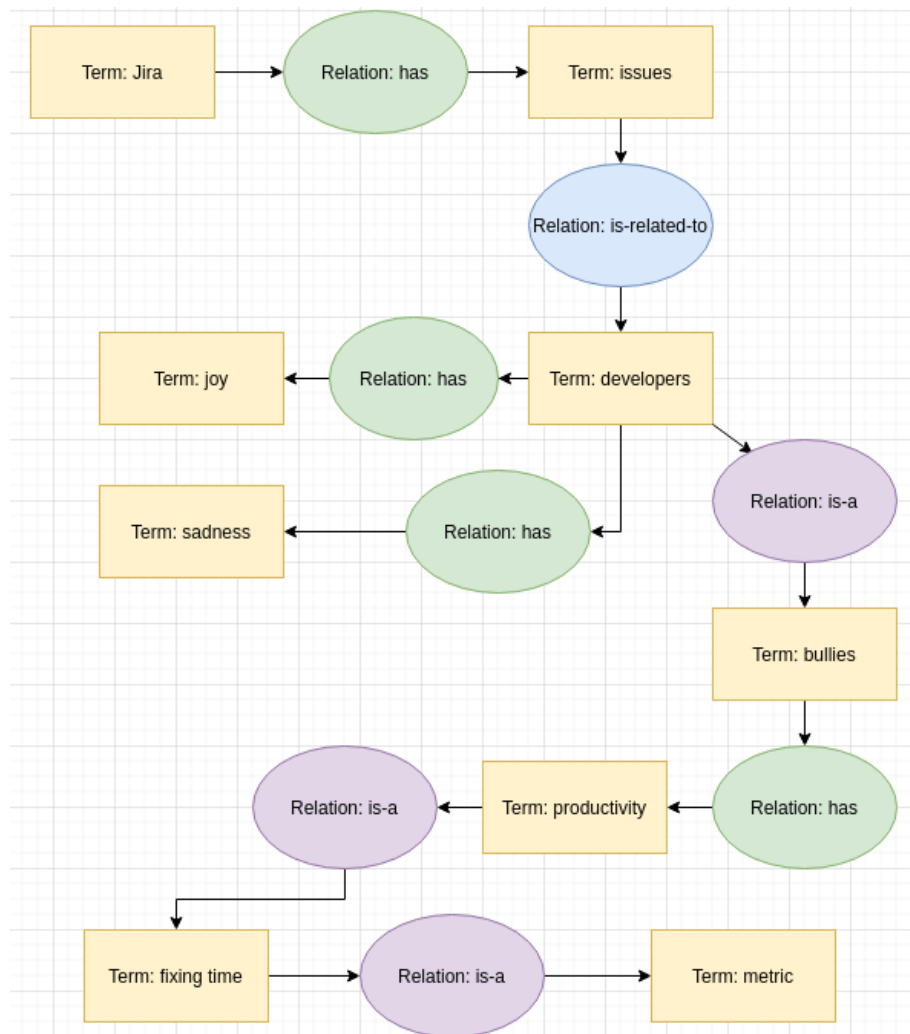


Figure 5: CG for (4)

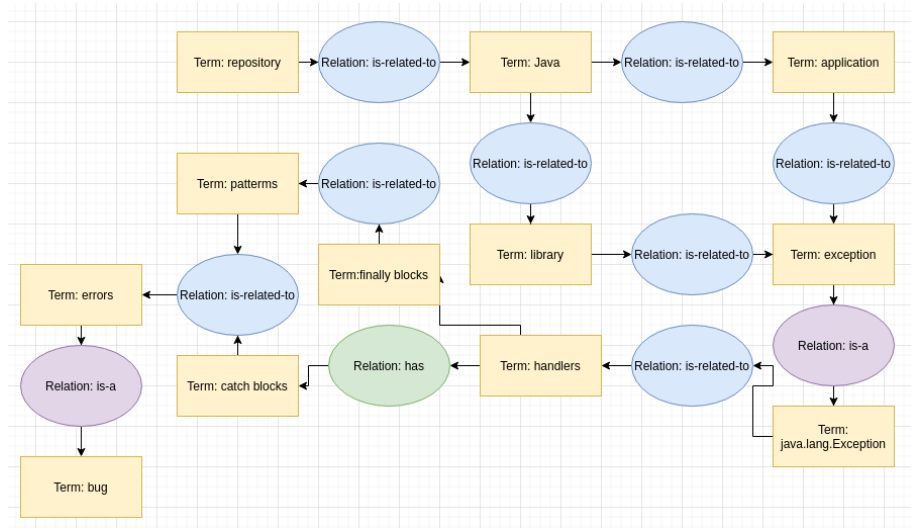


Figure 6: CG for (5)

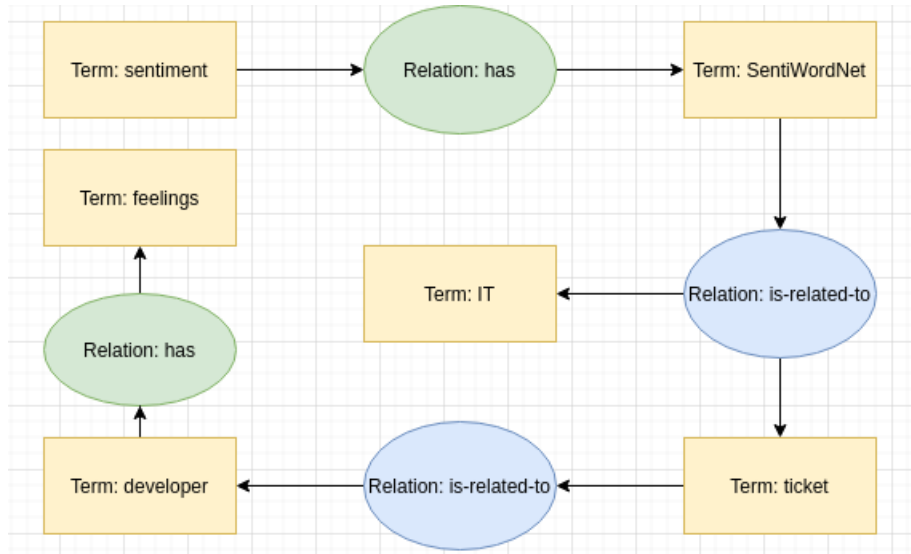


Figure 7: CG for (6)

Owing to such a graphical representation, path finding algorithms such as *Depth First Search* can be applied to traverse from a particular term to another for a set of relation edges. This will allow an author to draw inferences across multiple concepts with terms across different papers describe so as to formulate new research questions. An example of such a formulation

can be taken from section 1 where a traversal can start from the term *bug* through relations *is-a*, *is-related-to* and *has*. One would observe as a result of this traversal various inferences such as *bugs* being related to *patterns* and *anti-patterns*.

CHAPTER 3

RELATED WORK

As evidenced by literature, there exist several ways in which a conceptual graph can be constructed. Svetlena proposed one such way which used semantic role labelling knowledge bases such as *WordNet* and *VerbNet* and domain specific *syntactic patterns* a sentence can take (8). An interesting example is how Hasegawa et al create conceptual graphs to aid modelling requirements for software applications. They make use of *Cabocha* (9), which is a PoS tagger for the Japanese language to extract pertinent terms from a requirements specification and determine associations between these terms by encoding them into *co-occurrence vectors* and calculating the *cosine similarity* between them (9). Associations can be created between terms whose cosine similarities are higher (9).

It can be ordinarily understood that a domain expert can disambiguate terms pertaining to that domain. Ma'ayan et al explain how a team of such domain experts can be interviewed with the common purpose of creating *conceptual diagrams* (10). These diagrams provide a visual way of understanding relationships between abstract concepts (10). Another domain-centric approach comes up in the context of object oriented programming. Hines realized such a paradigm with the help of Sowa's techniques to construct a conceptual graph (11).

A use case of conceptual graphs is to make computing systems more intelligent. In one such instance brought forth by Nečaský, XML documents with different formats are *reverse engineered* by delineating its schema and mapping them to conceptual graphs (12). This allows for

the system to enforce a consistency when schemas are modelled (12). *Schematron* is a language which instead of relying on rule-based grammar instead makes use of conceptual models to model XML schemas (13). Similarly, UMLs can be understood better by assigning real-world semantics to a set of UML constructs (14). This is done by mapping these constructs to an ontology laid down by Mario Bunge, which is a standardized ontology for modelling information systems (14). Bouchakwa et al make use of conceptual graph formalisms to annotate images in a semantic context (15). (16) measures the plasticity of user interfaces which is a measure of the extent to which the usability of a UI is maintained while introducing variations to the context in which it is used. This paper enlists a three step process wherein first the situation is recognized, following which a remedy is computed and finally the remedy is executed (16). The descriptions of this process are visualized in the form of a CG (16).

A problem in our thesis is to determine terms which succinctly describe a research publication which in turn enables categorizing these papers. One such approach is to obtain key-phrases as provided by ACM's classification system (17). The idea is to compute a phrase-to-text relevance measure by using techniques such as cosine relevance score, pertinent characteristic of the probability of the term being generated and *CPAMF* which stores characteristics of conditional probabilities of fragments from various texts and phrases (17). Jo et al present a visualization based approach to build intuition about accessing papers rather than just extracting free form text (18). *Semantic linking systems* can be used which models a graph having document concepts as nodes and edges denote to what extent these concepts are related semantically. Using such a graph, the model proposed in (19) computes the conditional probability for a query

concept given certain values for concepts of a document.

Topic modelling is a technique which extracts latent topics over a corpus of documents. Latent Dirichlet Allocation (LDA) is one such topic modelling technique. In this technique, a topic is described as a proportion over words and each document as a distribution over a set of topics. Bayesian learning is used to determine such distributions. Owing to the intractability of computing these probability distributions, (20) develops a non-Bayesian approach called Additive Regularization of Topic Models which is based on regularized maximum likelihood estimation. Topic modelling are popular across domains such as in *law* (21) where legislative text is summarized and annotated with salient legal topics. (22) proposing a topic modelling methodology for *tweets* based on clustering called *ClusTop* which determines connected words and subsequently the topic which governs them using community detection techniques.

For the purpose of linking research papers, one proposed solution is to extract citations and textual content from these research papers and computing similarity by making use of SVM^{rank} (23). *Wikify* is a system which performs tasks such as keyword extraction and word-sense disambiguation to link documents with relevant Wikipedia pages (24). In the case of presentations, Erol et al present a retrieval technique which leverages image matching algorithms (25). Using this technique, presentations and particular slides in a presentation can be linked amongst thousands of them (25). For the purpose of knowledge management, *EROCS* posits an algorithm which maps predefined entities to text in a document (26). Linkage between documents which map to the same entities are linked (26). A use case in a legal setup is proposed by Shaffer

et al where court cases are linked to clauses in the US constitution (27). Paragraphs in legal documents are linked using a rule-based system and neural networks (27).

CHAPTER 4

TOOLS, NLP OPERATIONS AND TOPIC MODELLING TECHNIQUES

In this chapter we highlight various libraries/packages used along with the pertinent NLP operations and topic modelling techniques to infer concepts pertaining to our corpus of research papers.

4.1 Natural Language Processing Operations

4.1.1 Tokenization

Tokenization is the process of taking in a stream of characters and then slicing them into a set of predefined rules. Typically, these tokens are chopped in a way to represent words which have some semantic meaning (28).

4.1.2 Part of Speech Tagging

In linguistic studies, the process of part-of-speech tagging takes in raw text as input and assigns words in this raw text with some part-of-speech taking into account both its definition and the context within which it appears (29).

4.1.3 Term Expansion

Term expansion is a specific approach which is a part of a broader set of approaches under the umbrella of query expansion. Here, a term, specifically located in a query, can be replaced with synonyms or semantically similar terms to expand search results (30).

4.2 Topic Modelling

4.2.1 Latent Semantic Indexing

Latent Semantic Analysis is an information retrieval technique which discovers a latent semantic structure across terms belonging to a corpus of documents. In this technique, each document and term is expressed as a vector whose elements signify to what extent it can be fit into a particular structure (31).

4.2.2 Singular Value Decomposition for LSA

Consider a *term-document* matrix A . Here, each column of A represents a *document*. If a term i occurs x times in a document j then $A[i, j] = x$. A is an $m \times n$ matrix where m is the no. of *terms* and n is no. of *documents* (31).

Say, matrix $B = AA^T$ is a *document-document* matrix. $B[i, j] = b$ if documents i and j have b words in common. Likewise, $C = A^T A$ will be the *term-term* matrix where $C[i, j] = c$ implies that terms i and j have c documents in common. Here, both B and C are *square* and *symmetric* (31).

Owing to the above property of B and C , *SVD* can be performed on A . The general form of *SVD* for A is where S is matrix of *eigenvectors* of B and U is matrix of *eigenvectors* of C and Σ is a *diagonal matrix* (31):

$$A = S\Sigma U^T$$

Say, we wish to reduce the *rank* of A to k and call the matrix with reduced rank as A_k (31).

We can obtain it using the following:

$$A_k = S_k \Sigma_k U_k^T$$

where S_k , Σ_k and U_k are $m \times k$, $k \times k$ and $k \times n$ matrices and their product results in an $m \times n$ matrix (31).

Intuitively it can be said that the *k-participating vectors* in S and U are *hidden concepts* where the terms and documents participate and that they have a new representation in terms of these hidden concepts (31). Here, the terms are represented in terms of *row vectors* of $m \times k$ matrix (31):

$$S_k \Sigma_k$$

and documents are represented by the *column* of the $k \times n$ matrix (31):

$$\Sigma_k U_k^T$$

and the query can be encoded as the *centroid* of the vectors for the terms it contains (31).

Finally in order to rank documents with respect to some query q , the *cosine distance* can be used as below (31):

$$\frac{\mathbf{d}_i \cdot \mathbf{q}}{|\mathbf{d}_i||\mathbf{q}|}$$

Higher the cosine distance greater the similarity (31).

4.2.3 Relational Topic Modelling

Relational Topic Modelling (RTM) is a *hierarchical probabilistic model* of networks wherein each node contains attributes (32). At the time the authors came up with this model, the focus was primarily on document based networks such that this model can be used to model links between these documents as a binary random variable (32). Such a model is superior as preexisting models required a training set of observed links to predict new links whereas the RTM can do so by observing the attributes of a document and vice-versa (32). It is important to note that attributes here refer to the words contained in a document belonging to some fixed vocabulary (32).

In the RTM, each document is generated from some observed topic distribution as in the case of LDA (32). The links between documents are modelled as some binary random variable which is again dependent on this topic distribution. Due to the dependence of both words and links on topics, the content of a document is dependent on the link structure involving this document (32).

Here we introduce the joint posterior probability for modelling a link and how it can be computed using variational inference briefly.

4.2.3.1 Parameters of RTM

1. $\beta_{1:K}$: K multinomial parameters each of which describes a distribution on words (32).
2. α : A K -dimensional Dirichlet parameter (32).
3. Ψ : The link probability function which models a link between a pair of documents (32).
4. $w_{1:D,1:N}$: This notation denotes a set of documents where $w_{i,1:N}$ refers to the set of words contained in the i th document (32).
5. $y_{1:D,1:D}$: All links between a pair of documents. $y_{i,j} = 1$ if there is a link between documents i and j and is zero otherwise (32).

4.2.3.2 Word and link generation process for documents

1. For each document d :
 - (a) $\Theta_d | \alpha \sim \text{Dir}(\alpha)$ is a probability distribution which satisfies a Dirichlet distribution parameterized by α . Drawing from this distribution gives us some topic proportion (32).
 - (b) For each word $w_{d,n}$:
 - i. $z_{d,n} | \Theta_d \sim \text{Multi}(\Theta_d)$ is a probability distribution which satisfies a multinomial distribution parameterized by Θ_d . Here, assignment, say $z_{d,n}$ is drawn from the multinomial distribution Θ_d (32).
 - ii. For each topic chosen, say $z_{d,n}$, the word is chosen from the distribution of words on the topic chosen. From such a distribution $\beta_{z_{d,n}}$, a word, $w_{d,n}$ is chosen (32).

2. For a pair of document d, d' :

- (a) $y_{d,d'} | Z_d, z_{d'} \sim \Psi(\cdot | z_d, z_{d'}, \eta)$ satisfies the link probability function which is a probability distribution function (32). Here the probability distribution function modelling the link, Ψ depends on the pair of topic assignments and global regression parameters, η . Here $z_d = \{z_{d,1}, z_{d,2}, \dots, z_{d,n}\}$ (32)

4.2.3.3 Computing the link probability function

The authors look at the following possibilities to compute Ψ (32). These are:

$$\Psi_\sigma(y = 1) = \sigma(\eta^\top (\bar{z}_d \circ \bar{z}_{d'}) \mathbf{v}) \quad (4.1)$$

$$\Psi_e(y = 1) = \exp(\eta^\top (\bar{z}_d \circ \bar{z}_{d'}) + \mathbf{v}) \quad (4.2)$$

where $\bar{z}_d = \frac{1}{N_d} \sum_n z_{d,n}$ and the product between z_d and $z_{d'}$ denotes the Hadamard/element-wise product (32). η and \mathbf{v} denote the coefficients and intercept which parameterize the logistic regression used to model the per-document-pair binary variable (32). While Ψ_σ and Ψ_e use the same covariates, the former uses a sigmoid function (σ) and the latter the exponential function (\exp) (32).

4.2.3.4 Inference, estimation and prediction

Computing the posterior for hierarchical Bayesian models is intractable and therefore arises the need to approximate it. The authors turn to *variational inference* whose methods assume a

family of distributions and the one with the lowest difference in relative entropy with respect to the original posterior is selected. The fully-factorized family is selected where topic proportions and topic assignments are considered as independent random variables (32).

$$q(\Theta, Z | \gamma, \Phi) = \prod_d \left[q_\theta(\theta_d | \gamma_d) \prod_n q_z(z_{d,n} | \Phi_{d,n}) \right] \quad (32)$$

In the above equation, γ refers to the variational Dirichlet parameters for each document and Φ are multinomial variational parameters for each word in each document (32).

Minimizing the relative entropy is the same as maximizing the Jensen's lower bound on the marginal probability across all observations. This quantity is known as the evidence lower bound (32):

$$\begin{aligned} \mathcal{L} = \sum_{(d_1, d_2)} E_q[\log p(y_{d_1, d_2} | z_{d_1}, z_{d_2}, \eta, \nu)] + \sum_d \sum_n E_q[\log p(z_{d,n} | \theta_d)] + \sum_d \sum_n E_q[\log p(w_{d,n} | \beta_{1:K, z_{d,n}})] \\ + \sum_d E_q[\log p(\theta_d | \alpha)] + H(q) \end{aligned} \quad (4.3)$$

(d_1, d_2) denote all document pairs in the corpus and $H(q)$ is the entropy of the distribution q (32). For more efficient computation, a pair of documents is treated as observed iff $y_{d_1, d_2} = 1$ (32). The intuition offered here is that treating $y_{d_1, d_2} = 0$ as unobserved instead is more

truthful to the fact that a link could exist but does not ascertain that a link doesn't (32).

Now the aim is to compute each term in the above equation. The first term is:

$$\sum_{d_1, d_2} \mathcal{L}_{d_1, d_2} \equiv \sum_{(d_1, d_2)} \mathbb{E}_q[\log p(y_{d_1, d_2} | z_{d_1}, z_{d_2}, \eta, \nu)] \quad (4.4)$$

Using a first order approximation and implying the function to only depend on $\bar{z}_{d_1} \circ \bar{z}_{d_2}$, the equation can be rewritten as (32):

$$\mathcal{L}_{d_1, d_2} = \mathbb{E}_q[\log \Psi(\bar{z}_{d_1} \circ \bar{z}_{d_2})] \approx \log \Psi(\mathbb{E}_q[\bar{z}_{d_1} \circ \bar{z}_{d_2}]) = \log \Psi(\bar{\pi}_{d_1, d_2})$$

where $\bar{\pi}_{d_1, d_2} = \bar{\Phi}_{d_1} \circ \bar{\Phi}_{d_2}$ and $\bar{\Phi}_d = \mathbb{E}_q[\bar{z}_d] = \frac{1}{N_d} \sum_n \Phi_{d, n}$. The link probability functions can be now expressed as (32):

$$\mathbb{E}_q[\log \Psi_\sigma(\bar{z}_{d_1} \circ \bar{z}_{d_2})] \approx \log \sigma(\eta^\top \bar{\pi}_{d_1, d_2} + \nu) \quad (4.5)$$

$$\mathbb{E}_q[\log \Psi_e(\bar{z}_{d_1} \circ \bar{z}_{d_2})] = \eta^\top \bar{\pi}_{d_1, d_2} + \nu$$

Once the expectations have been obtained, the ELBO is to be optimized with respect to the variational parameters γ and ϕ (32). The update for the variational multinomial comes out to be (32):

$$\Phi_{d, j} \propto \exp \left\{ \sum_{d' \neq d} \nabla_{\Phi_{d, n}} \mathcal{L}_{d, d'} + \mathbb{E}_q[\log \phi_d | \gamma_d] + \log \beta_{\cdot, w_{d, j}} \right\} \quad (4.6)$$

The update part of the above equation $\nabla_{\Phi_{d,n}} \mathcal{L}_{d,d'}$ depends on the link probability function (32).

$$\nabla_{\Phi_{d,n}} \mathcal{L}_{d,d'} = (\nabla_{\bar{\pi}_{d_1,d_2}} \mathcal{L}_{d,d'}) \circ \frac{\bar{\Phi}_{d'}}{N_d} \quad (4.7)$$

The above equation will cause a document's latent topic assignments to tilt towards its neighbouring document's topic assignments (32). The magnitude of this depends solely on $\bar{\pi}_{d,d'}$ which is a measure of how close the two documents are. The following gradients of the link probability functions are as below (32):

$$\nabla_{\bar{\pi}_{d,d'}} \mathcal{L}_{d,d'}^{\sigma} \approx (1 - \sigma(\eta^T \bar{\pi}_{d,d'} + \nu)) \eta$$

$$\nabla_{\bar{\pi}_{d,d'}} \mathcal{L}_{d,d'}^{\epsilon} = \eta$$

The contribution of the word evidence in Equation 4.6 $\log \beta_{\cdot, w_{d,j}}$ can be computed by taking element-wise logarithm of the $w_{d,j}$ th column of the topic matrix β . The contribution of the document's latent topic proportions to the update is given below (32):

$$E_q[\log \theta_d | \gamma_d] = F(\gamma_d) - F(\sum \gamma_{d,i})$$

where F is the digamma function. The update for variational Dirichlet parameters γ is given as (32):

$$\gamma_d \leftarrow \alpha + \sum_n \phi_{d,n}$$

For the purpose of fitting the model, maximum likelihood estimates for multinomial topic vectors $\beta_{1:K}$ and link function parameters η, \mathbf{v} are to be determined (32). As this is again an intractable problem, variational expectation-maximization is performed where ELBO is optimized with respect to variational distribution and model parameters (32). Here, the update for topics matrix β is given as (32):

$$\beta_{k,w} \approx \sum_d \sum_n 1(w_{d,n} = w) \phi_{d,n,k}$$

To fit parameters η and \mathbf{v} of the logistic regression function given in Equation 4.1, gradient-based optimization is used (32). Gradient descent equations are as below:

$$\begin{aligned} \nabla_{\eta} \mathcal{L} &\approx \sum_{d_1, d_2} [y_{d_1, d_2} - \sigma(\eta^T \bar{\pi}_{d_1, d_2} + \mathbf{v})] \bar{\pi}_{d_1, d_2} \\ \frac{\delta}{\delta \mathbf{v}} \mathcal{L} &\approx \sigma_{(d_1, d_2)} [y_{d_1, d_2} - \sigma(\eta^T \bar{\pi}_{d_1, d_2} + \mathbf{v})] \end{aligned}$$

Once the model has been fit, the goal now is to make predictions about links given words and vice-versa (32). For link prediction, a new document with words not part of the training set is given and links are to be computed (32). This probability distribution is expressed as:

$$p(y_{d,d'} | w_d, w_{d'}) = \sum_{z_d, z_{d'}} p(y_{d,d'} | \bar{z}_d, \bar{z}_{d'}) p(z_d, z_{d'} | w_d, w_{d'})$$

Our goal now is to find variational parameters which optimize the ELBO for evidences such as words and links from the training set and words from the test document (32). The probability can now be approximated as (32):

$$p(y_{d,d'}|w_d, w_{d'}) \approx E_q[p(y_{d,d'}|\bar{z}_d, \bar{z}_{d'})] \quad (4.8)$$

Similarly word prediction can be computed using the same variational technique as below (32):

$$p(w_{d,i}|y_d) \approx E_q[p(w_{d,i}|z_{d,i})] \quad (4.9)$$

4.2.4 Expected Entropy Loss

The *Expected Entropy Loss* (EEL) is an algorithm which is used to extract the most relevant attributes for a set of documents (33). This algorithm works by ranking attributes based on how well it describes a given category. The indicator which is used to quantify this ranking is the attribute's entropy for that category. Consider the example of a term, say "*uml*". This term is more likely to belong to the category of "*Modelling*" as opposed to "*Version Control Systems*".

More formally, entropy is a measure of the uncertainty associated with an event and is expressed in terms of a discrete set of probabilities over some event space where x_i represents some event.

Entropy is computed as below (33):

$$e(X) = \sum_{n=1}^n \Pr(x_i) \log(\Pr(x_i))$$

Consider the following variables - C which corresponds to some event denoting whether an application belongs to a certain category and a be the event that a document contains a certain term. We can now define the following equations (33):

$$\Pr(C) = \frac{\text{Number of related documents}}{\text{Number of documents}}$$

$$\Pr(\bar{C}) = 1 - \Pr(C)$$

$$\Pr(a) = \frac{\text{Number of documents with term } a}{\text{Number of applications}}$$

Here, $\Pr(C)$ is the probability that for a certain category, the document will belong to that category and $\Pr(a)$ is the probability for each attribute, that a document will contain that attribute (33).

$$\Pr(\bar{a}) = 1 - \Pr(a)$$

$$\Pr(C | a) = \frac{\text{Number of related documents with term } a}{\text{Number of documents with term } a}$$

$$\Pr(\bar{C} | a) = 1 - \Pr(C | a)$$

$$\Pr(C | \bar{a}) = \frac{\text{Number of related documents without term } a}{\text{Number of documents without term } a}$$

$$\Pr(\bar{C} | \bar{a}) = 1 - \Pr(C | \bar{a})$$

The prior entropy which corresponds to the distribution of documents for a category is computed as below (33):

$$e(C) = -\Pr(C)\log(\Pr(C)) - \Pr(\bar{C})\log(\Pr(\bar{C}))$$

The posterior entropy calculates the probability for a given term being categorized with a certain category (33).

$$e_a(C) = -\Pr(C | a)\log(\Pr(C | a)) - \Pr(\bar{C} | a)\log(\Pr(\bar{C} | a))$$

For the case where the attribute is not to be categorized with a certain category, we have the below equation (33).

$$e_{\bar{a}}(C) = -\Pr(C | \bar{a})\log(\Pr(C | \bar{a})) - \Pr(\bar{C} | \bar{a})\log(\Pr(\bar{C} | \bar{a}))$$

Therefore it can be said that the *expected posterior entropy* can be calculated as (33):

$$\text{EPE}(C, a) = e_a(C)\Pr(a) + e_{\bar{a}}(C)\Pr(\bar{a})$$

Finally, the entropy loss is computed as follows (33):

$$\text{EEL}(\mathbf{C}, \mathbf{a}) = \mathbf{e}(\mathbf{C}) - \text{EPE}(\mathbf{C}, \mathbf{a})$$

The EEL value can be computed for every term for each category where the higher the value, more the distinction when it comes to categorizing the term (33).

4.3 Important tools used

In this section, we list down some key tools/libraries to implement key aspects of our application.

4.3.1 Natural Language Toolkit

Natural Language Toolkit (*NLTK*) is a package which enables performing pertinent NLP tasks such as classification, tokenization, stemming, tagging, parsing and semantic reasoning (34).

4.3.2 WordNet

To put it simply, “WordNet® is a large lexical database of English” (35). It uses what is known as synsets to group words which are semantically similar. One may think of it as a database representing a thesaurus but it goes further to store what sense the words are used in (35).

It stores relations between words which realizes a word being a hypernym/hyponym of another word (35). What this denotes is whether there exists a type-of relationship(hypernym) between words or vice-versa (hyponymy). An example is, “*an eagle is a hypernym of a bird*” or “*a*

furniture is a hyponym of a bunk bed".

This database helps us achieve the objective of term expansion as we shall see in a subsequent chapter.

4.3.3 PdfBox

Apache's PdfBox is a Java based library which helps in the creation, manipulation and extraction of textual content from PDF documents (36).

4.3.4 flask

`flask` is a Python-based web application framework which is used to create mappings between URLs and functions and allows an application to be deployed in a web server. It does so without forcing a developer to choose specific libraries which it may depend on (37).

4.3.5 marshmallow

`marshmallow` is a Python-based library which performs operations such as Object Relation Mapping (ORM) and Object Database Mapping (ODM). More concisely, it is used to create schema on which input data can be validated. This schema data can be serialized to through input data coming in as a request object from a HTTP call. These schema objects can be deserialized back into standard forms such as JSON (38).

4.3.6 scikit-learn

`scikit-learn` is a Python-based library which provides functions to perform supervised and unsupervised machine learning. It provides models out of the box along with utilities for data pre-processing and evaluation (39).

4.3.7 science-parse

`science-parse` is a utility written in the Scala language which parses published articles and extracts from them pertinent information such as the abstract, author list, references and the text pertaining to individual sections in the chosen article (40).

4.3.8 pymongo

`pymongo` is a Python-based library which provides the functionality of connecting to a MongoDB backend and performing database-specific operations such as create, read, update and delete (*CRUD*) operations (41).

CHAPTER 5

OUR CORPUS

The corpus we chose are an assortment of research papers published in the Mining Software Repositories’ conferences. This field analyzes data from software repositories and reports insightful results concerning software projects and systems (42). We start off from a corpus of 603 research papers whose textual content is extracted using Scala’s Apache’s PdfBox package. Owing to the structured manner a research paper is written, it becomes easier to build *regular expressions* to extract content so as to perform some interesting analysis on its content.

5.1 Sections of a research paper

Commonly, a research paper has the following sections - *Abstract, Introduction, Motivation, Future Work, Conclusion, References*. We perform a manual inspection of 50 papers to look specifically for sections and check for the occurrences of these sections using Python’s `re` package. The results we obtain are shown in Table I.

5.2 Authors

We analyze for our corpus, the authors of the research papers. We particularly look at authors with the *most publications* and the *most citations*. To this end, AllenAI’s `science-parse` package is used which extracts author names. Citations are extracted using Python’s `refextract` package. Metadata pertaining to authors are tabulated in Table II.

TABLE I: Paper Sections' Metadata

Term	No. of papers occurring in
Abstract/ABSTRACT	308/314
Acknowledgements/ACKNOWLEDGEMENTS	22/35
Approach/APPROACH	139/64
Background/BACKGROUND	19/80
Case Study/CASE STUDY	62/56
Conclusion/Conclusions/CONCLUSION/CONCLUSIONS	28/30/317/182
Discussion/DISCUSSION	68/169
Future Work/FUTURE WORK	25/128
Introduction/INTRODUCTION	98/546
Methods/METHODS	121/22
Motivation/MOTIVATION	48/16
Problem Statement/PROBLEM STATEMENT	4/1
References/REFERENCES	55/550
Related Work/RELATED WORK	34/276
Results/RESULTS	222/240
Summary/SUMMARY	69/37

TABLE II: Authors Metadata

Indicator	Count
Total no. of authors extracted	1239
Max. publications for an author	20
Median no. of publications for an author	1

5.3 Research questions

Research questions (RQs) form a pertinent part of research papers as they enlist the questions which the researcher plans on investigating. We posit that a typical research question is as *"RQ1: What is the correlation between no. of commits and hours worked by a developer?"*. We apply the following regex $((RQ|rq)\backslash s*[0-9])([\^.?!])+\backslash ?$ to extract RQs and there were a total of 360 papers for which we could do so. There were a total of 5 papers that had questions which started with a 'Q' or did not have a '?' as shown in Table III.

TABLE III: Research Questions Metadata

Indicator	Count
Research questions extracted	679
Max RQs for a paper	7
No. of papers with RQs	360
No. of papers with undetected RQs	5

5.4 References

References provide a way to cite other articles whose content has been used in a research paper. We rely on the structure of a research article and look specifically for the strings *"REFERENCES"* and *"References"* instead of *"references"* or the far more unlikely *"reFeRences"*. All text following the string are then extracted and upon observation we see that an example of a typical reference is *"[24] Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. 2014."*

TABLE IV: Most common nouns occurring in research questions

Noun	Frequency
code	70
developers	61
time	43
different	40
models	39

TABLE V: Citations Metadata

Indicator	Count
Total papers	592
Total citations extracted	11271
Min. citations for a paper	1
Max. citations for a paper	81
Median of citations for a paper	15
Max. citations for an author	217
Median of citations for an author	1
No. of authors across all citations	11208

A Study of External Community Contribution to Open-source Projects on GitHub. In MSR. ACM, 332–335.” and we employ the regex `\[[0-9]+\]` to split at. We obtain the following statistics with respect to references as shown in (?)

5.5 Content of the paper

In this section, we analyze the most frequently occurring terms and the no. of terms occurring in a paper. Terms are tokenized by removing special characters and splitting the residual text at white spaces. The results are displayed in Table VI, Table VII. *Tables* and *figures* provide a way for a paper to report results. We analyze the no. of papers which make use of them in Table VIII.

TABLE VI: Top 5 term frequencies

Term	Frequency across all papers
software	19752
code	18896
data	11476
number	8777
source	8640

TABLE VII: Top 5 papers by no. of words

Paper Title	No. of terms
The Maven Dependency Graph: a Temporal Graph-based Representation of Maven Central	27092
An Empirical Study on Android-related Vulnerabilities	7544
Data-Driven Search-based Software Engineering	7470
Deep Learning Similarities from Different Representations of Source Code	7420
Analyzing Requirements and Traceability Information to Improve Bug Localization	7021

TABLE VIII: Miscellaneous paper metadata

Indicator	Frequency
Papers with reference to a GitHub URL	201
Papers with Figure/FIGURE	514/1
Papers with Table/TABLE	496/173

CHAPTER 6

SOLUTION

Our primary objective for this thesis is to create a conceptual graph. We identify the following concepts to be realized by our conceptual graph:

1. *paper*: Here, a paper denotes an article published in various editions of the *Mining Software Repositories*' conferences.
2. *term*: A term is a lexical token present in a paper. Consider a paper P1 with text "*I am a boy*". In this case, our $term_set = \{ "I", "am", "a", "boy" \}$ consists of elements each of which has some semantic meaning in the domain of software engineering.
3. *category*: A category in the context of our application is a semantic umbrella for related papers and terms these papers contain. An example of a category is "*Version Control Systems*" which could be used to categorize a paper titled as "*What is the Gist? Understanding the Use of Public Gists on GitHub*". For anyone familiar with version control systems, GitHub is a very popular web client which employs the *git* VCS. On a similar note, terms which these papers contain can be categorized. A point to note is that there exists a *many-to-many* relation between a category and a paper or a category and a term.
4. *relation*: A relation connects two terms either across two papers or within the same paper which are semantically related. In the context of our application, a relation is represented

as a phrase. An example of a relation is, the term “*object*” is related to “*class*” where the relation is “*is related to*”. In our implementation, a relation is *bi-directional* in nature.

5. *Relation inference strategy*: A relation could be added using multiple strategies. These include the following:

- (a) *GROUND_TRUTH*: This allows a domain expert to manually add relations between terms.
- (b) *TERM_EXPANSION*: Through this strategy, synonyms for already related terms are obtained and new relations using synonyms are added.

6.1 Extracting textual content from papers

Research papers like any other textual document has content with special characters, stop words, numbers and other non-alphabetical characters etc. which do not make much semantic sense. A preliminary step is to preprocess this content by using precise regular expressions, language specific stop words and specifying common delimiters about which a sentence can be tokenized into terms belonging to some fixed vocabulary. We enlist the steps taken to tokenize the content of a paper:

1. Extract textual content of a paper using Apache’s **PdfBox** library.
2. Extract the title of the paper, remove special characters, numbers, null valued characters from the textual content of the paper and split the residual text using whitespaces as a delimiter.
3. Extract sentences from a paper by splitting residual text at full stops.

6.2 Filtering terms in a paper

In spite of the previous step, terms often had special characters which required the need to specify their respective Unicode values so as to remove them. To fix this issue, we build two lists - *ground truth terms* and *super tech ground truth terms*. As we specify in a later chapter, we provide the end user with two components - *Build ground truth terms* and *Build super tech ground truth terms*. For the former, the user builds a subset of terms from the original list of terms across the entire corpus and for the latter, a specialized list of terms from the ground truth terms is built. This also allows for greater technical focus when it comes to ascertaining candidate terms for our conceptual graph.

6.3 Contextualizing terms in a paper

Now it may so happen that a term in a paper may appear in varying contexts in spite of confining our realm of terms to that of empirical software engineering. An apt example is that of the term "*Abstract*" which can either denote the summary of a paper or it may appear in the phrase "*Abstract Syntax Tree*" which is the syntactic representation of some source code in the form of a tree. To allow the user greater understanding of all possible usages of a term, we list out all sentences where the term appears. This enables better categorization of such terms along with encapsulating all possible contexts when adding relations pertaining to these terms.

6.4 Defining relations

Relations provide a way of introducing semantics when a system is to understand real word knowledge. This was all the more important within the database community. Backed by research done in the fields of logic, linguistics and cognitive psychology semantic relationships

were categorized as *inclusion*, *possession*, *attachment*, *attribution*, *antonym*, *synonym* and *case*

(7). Diagrammatically this taxonomy is as follows:

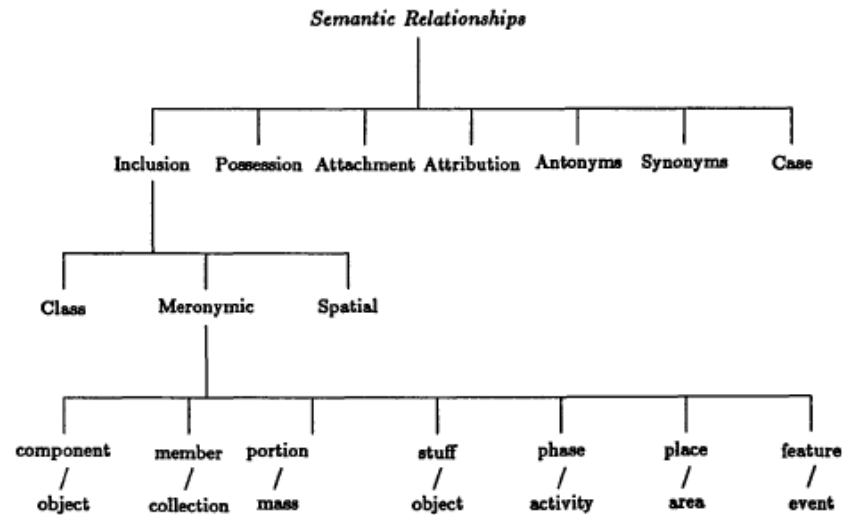


Figure 8: Semantic relationships (7)

The semantic relations can be defined as:

1. *Inclusion*: This relationship describes a situation when an entity type comprises or contains another entity (7). An example is "Car *is a* type of Vehicle".
 - (a) *Class*: This represents a typical *subtype/supertype* relationship (7).
 - (b) *Meronymic*: Here, the *part-of* relationship is denoted (7).
 - (c) *Spatial*: Spatial relations describe situations where one object is surrounded by another but is not part of the thing that surrounds it (7). E.g. "Buyer *is in* the candy shop."
2. *Possession*: This denotes the ownership relationship (7). E.g. "Person *possesses* a job."

3. *Attachment*: In this type of relationship, one entity type is connected/joined to another type (7). E.g. "The door knob *is attached to* the mug."
4. *Attribution*: An attribution specifies some property that an entity exhibits (7). E.g. "Bird *has* wings."
5. *Antonyms*: Antonyms can occur in attributes, entities, or relationships alike. Two entity/attributes/relationships types will be mutually exclusive if they are antonyms (7). An example is "Borrowers *borrow* books" and "Borrowers *return* books".
6. *Synonyms*: As with antonyms, synonyms also characterize entities, their attributes and the relationships they may occur in (7). E.g. "Worker *performs* on the job." and "Employee *performs* on the job".

We start off by entering in preliminary relations into the system such as *is-a* (inclusion), *part-of* (meronymy), *has* (attribution), *use* (attachment), *attached-to* (attachment) and display it in a list box so that the user has some relation templates to relate two terms. To this end, we built an *add relations* component which we detail in 4.

6.5 Building the conceptual graph

Once we have defined our concepts, associations between these concepts have to be created.

We identify the following associations:

1. Every term can have one or more categories.
2. Every paper can have one or more categories.
3. A pair of terms across papers can have one or more relations.

We build Angular components which allows the user to create these associations. A usability issue came up when adding relations between two terms. We initially built a component which had two auto-complete elements where the titles of both the papers would be specified and two drop-downs which displayed the list of terms of each paper as shown below.

The image shows a web form titled 'Add term relations' with two identical sections for selecting paper titles and terms. Each section includes a text input for the paper title, a blue 'Fetch terms for paper' button, a dropdown menu for terms, and a text area for context. At the bottom, there are two green buttons: 'Add term relation' and 'Reload relations'.

Enter the title of the paper from which the first term is to be related

Fetch terms for paper

Terms

Context in which the term is used

Enter the title of the paper from which the second term is to be related

Fetch terms for paper

Terms

Context in which the term is used

Select/Enter the relation:

Add term relation

Reload relations

Figure 9: Add term relations component

This led to extremely long drop-downs of which the user didn't have a more widespread view of. Subsequently, we visualized our graph as an adjacency matrix where the x-axis and y-axis corresponded to the terms of the two selected papers. On a large enough screen, say a 45 inch screen, this allows the user to have a view large enough to see multiple relations across

terms. Each cell of the matrix is a button whose colour would change from red to green in the case where a relation between the two terms have been defined.

CHAPTER 7

ARCHITECTURE

In this chapter, we explain in detail the design of our application which achieves the task of building a conceptual graph.

7.1 Application design

The application we build follows the *model-view-controller* (MVC) architectural pattern. We build the view component using *Angular* which is a JavaScript-based framework used for building responsive user interfaces. The model and the controller components are a part of a Python-based web application which exposes RESTful endpoints to carry out pertinent CRUD operations on a *Mongo* based database. The user interface project is **grouse-ui** and the Python-based back end project is **grouse-server**.

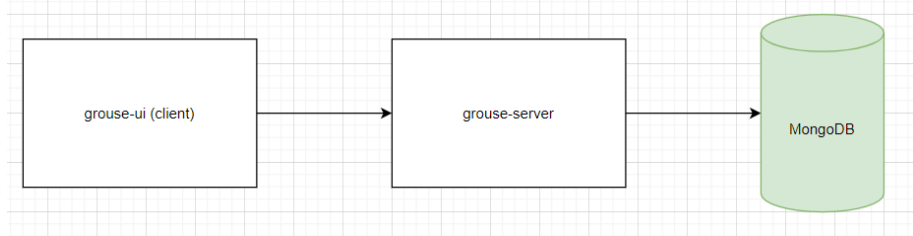


Figure 10: *GROUSE* flow diagram

7.1.1 grouse-ui: The user interface

The various components in this application are described below. A component is a unit which groups elements to create a view (43).

1. *Add category*: This component enables the user to define a category.

Enter the category

Add category

Figure 11: Add category component

2. *Add paper term category*: This component is required to assign a category for a selected term in a chosen paper. The user execution flow first requires the paper to be chosen from an auto-complete enabled input field after which the terms specific to the paper are fetched and chosen.

Search for a paper

Enter the term set option

Ground truth

Fetch terms for paper

Terms cointrader

Context in which the term is used

```
net
cpvrlab/ImagePlay pipelka/roboTV
crowdcode-de/KissMDA plt-tud/r43ples
```

Categories Modelling

Add term category Delete term category

Reload categories

Figure 12: Add paper term category component

3. *Add paper category*: Similar to the previously mentioned component, this component allows the user to specify a category for a paper.

Enter the paper title :

Select the category for the paper

Modelling ▼

Add category for paper Delete category for paper

Reload categories

Figure 13: Add paper category component

4. *Add relation* : This component facilitates defining a relation as a template which can be added to relate two terms in the future.

Enter the relation

Add relation Delete relation

Figure 14: Add relation category component

5. *Merge categories*: The use case of this component is so that categories which could possibly be semantically similar are merged into a single category. For example, say the user defines categories such as "*General*" and "*Mainstream*". There could then be a need to store just one of these categories. The user can then decide to merge "*General*" into "*Mainstream*" or vice-versa. To this end, the component displays a *select* component to select a category to be merged into and a *multi-select* component to select categories to merge.

Select categories to be merged

Modelling
 Version Control Systems
 Unified Modelling

Select category to be merged into

Unified Modelling

Merge categories

Reload categories

Figure 15: Merge categories component

6. *View relations*: Using this component, one can view all relations defined for some term. It displays a table which consists of the relation, both the primary and secondary term which are part of the relation and the papers to which the respective terms belong to. A button is provided for the option to delete the relation.

Enter the term set option

Ground truth

Search for a paper

Beyond GumTree: A Hybrid Approach to Generate Edit Scripts

Fetch terms for paper

Terms applied

Fetch all terms relations

Primary Paper title	Primary term	Relation	Secondary term	Secondary Paper title	Relation inference strategy	Delete relation
Beyond GumTree: A Hybrid Approach to Generate Edit Scripts	applied	is-a	archive	Software Bertillonage: Finding the Provenance of an Entity	GROUND_TRUTH	Delete relation

Figure 16: View relations component

7. *Build concept matrix*: In this component, the user first chooses a couple of papers by specifying their respective titles. Then, the *Build conceptual graph* button is clicked. What this creates is a adjacency matrix like structure with the terms of both the first and second paper. Here the user can click a button corresponding to a pair of such terms which in turn opens up to a modal to add a relation between the two terms. The modal also consists of a *text-area* element which displays the sentences in which the terms appear to show context.

Enter the title of the paper from which the first term is to related

Beyond GumTree: A Hybrid Approach to Generate Edit Scripts

Enter the title of the paper from which the second term is to be related

Software Bertillonage: Finding the Provenance of an Entity

Build term matrix

Enter the term set option

Ground truth

#	*	abstract	algorithm	apache	api	applied
	abstract	abstract	algorithm	apache	api	applied
	access	abstract	algorithm	apache	api	applied
	accumulation	abstract	algorithm	apache	api	applied
	achieve	abstract	algorithm	apache	api	applied

Figure 17: Build concept matrix component

8. *Build ground truth terms*: The purpose of this component is to create a subset from all terms across the corpus of papers. In this component, there are two elements: one from

which the user can select a term from a possible ground truth (adds it to the ground truth) and another from which the added ground truth terms can be viewed. Clicking a term in the ground truth adds it back to the list of possible ground truth terms. As there were a total of *93607* terms across all papers, rendering all of them causes the browser's memory to be exhausted thereby leading to the page hanging. To fix this, we make use of Angular's *Component Development Kit* (CDK) which provides the virtual scrolling functionality. This kit consists of elements which only react to scrolling events thereby rendering only what fits on-screen (44).

This ground truth of terms is reflected when it comes to adding a relation between two terms or categorizing these terms. With this, terms which are used to build conceptual graphs are carefully curated thereby removing those which contain special characters and do not make any semantic sense.

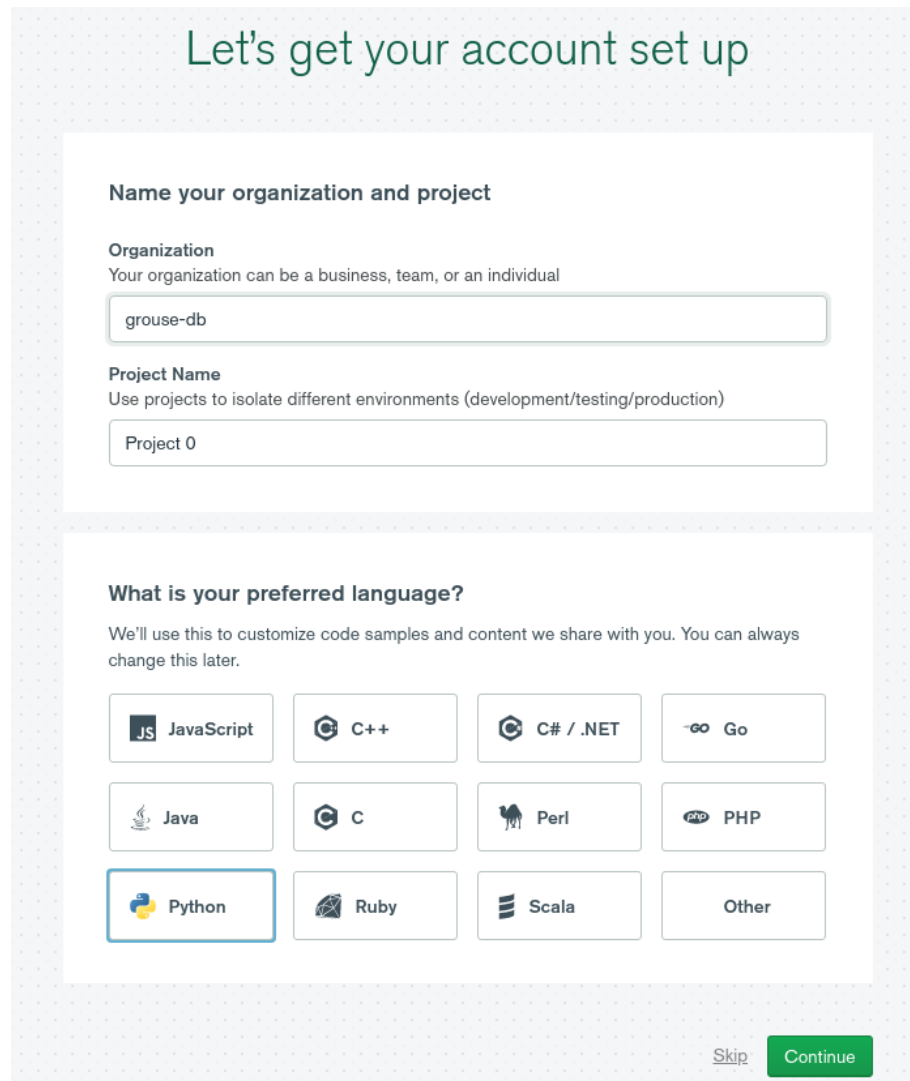
9. *Downloads*: Using this component, the user can download data such as *relations*, *categories* and *relations between terms*. The downloaded data is in the *JSON* format.

The service class files - `terms.service.ts`, `paper.service.ts` with RESTful endpoints defined in `grouse-server`. It is important to note the domain in which `grouse-server` and initialize the `baseURL` with this domain.

7.1.2 grouse-db: The database

To this end, we make use of MongoDB's Atlas to set up a MongoDB cluster. Atlas is a cloud-based database service fully managed by AWS. In order to set up such a cluster, we follow the steps below:

1. Create an account on the following link: <https://www.mongodb.com/cloud/atlas/signup>
2. Specify the organization and project name along with the preferred language.



Let's get your account set up

Name your organization and project

Organization
Your organization can be a business, team, or an individual

grouse-db

Project Name
Use projects to isolate different environments (development/testing/production)

Project 0

What is your preferred language?
We'll use this to customize code samples and content we share with you. You can always change this later.

JavaScript C++ C#/.NET Go

Java C Perl PHP

Python Ruby Scala Other

[Skip](#) [Continue](#)

Figure 18: Project, organization and language specification

3. Specify the cloud service provider and the cluster. It is best to choose an availability zone close to your physical location.

1 Select a cloud provider



2 Select a cluster configuration

M10 Cluster (Dedicated)

- 10GB+ storage
- No downtime scaling
- Network isolation
- Real-time performance metrics

M2 Cluster (Shared)

All M0 features plus:

- 2GB storage
- Daily backup snapshots

FREE FOREVER

M0 Cluster (Shared)

- 512MB storage
- Auto-healing cluster
- End-to-end encryption
- Role-based access control

Feature	Your Cluster
Cloud Provider	AWS
Country / Region	🇪🇺 Europe / Ireland (eu-west-1)
Cluster Tier	M0 Cluster (Shared)
RAM / Disc Size / vCPU	Shared / 512 MB / Shared
Backups	Only available for M2+ clusters See our backup solutions
Pricing	Starting at FREE

[Configure My Cluster](#)
[Build M0 Cluster](#)

Figure 19: Cloud service provider and cluster configuration

4. Specify the authentication mechanism, the IP address which is to be allowed to authenticate and the environment from which the connection is to be made.

1 How would you like to authenticate your connection?

Your first user will have permission to read and write any data in your project.

Username and Password
Certificate

Create a database user using a username and password. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password.

Username

Password

Enter username
Enter password
Create User

2 Where would you like to connect from?

Enable access for any network(s) that need to read and write data to your cluster.

My Local Environment
Use this to add network IP addresses to the IP Access List. This can be modified at any time.

ADVANCED
Cloud Environment
Use this to configure network access between Atlas and your cloud or on-premise environment. Specifically, set up IP Access Lists, Network Peering, and Private Endpoints.

3 Set your network security with any of the following options

Only an IP address you add to your Access List will be able to connect to your project's clusters.

IP Address	Description		
Enter IP Address	Enter description	Add Entry	Add My Current IP Address

Back
Continue

Figure 20: Enabling data access

- As we make use of the pymongo library to connect to Atlas, we choose the *"Connect your Application"* option as shown below:

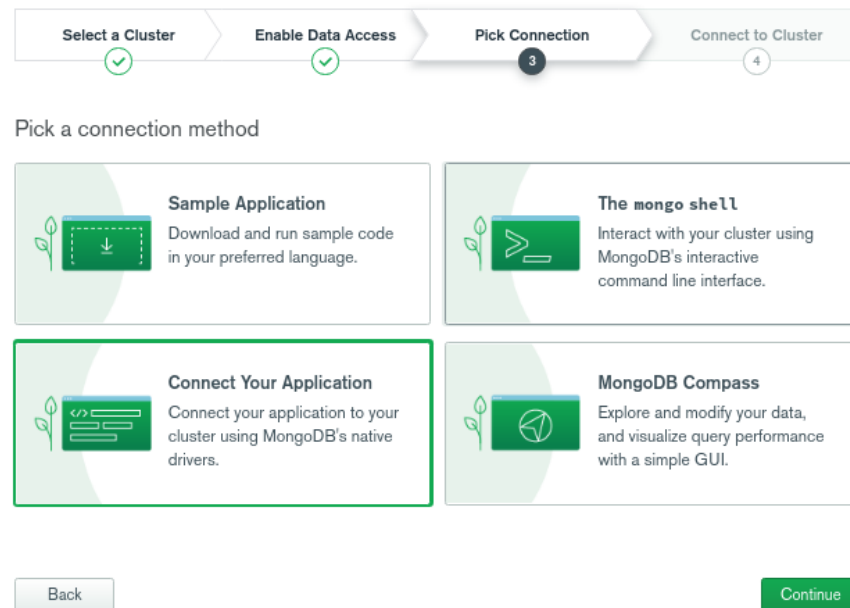


Figure 21: Picking the connection method

6. Copy the following code along with replacing the required parts with your username and credentials required for authenticating the connection:

NOTE

You will need your database username and password you created in the previous step to connect to your cluster.

- 1 Choose your driver

Python ▼
- 2 Add your connection string into your application code


```
client = pymongo.MongoClient("mongodb+srv://fhsdjfh:<password>@cluster0.1knwk.mongod
b.net/<dbname>?retryWrites=true&w=majority")
db = client.test
```

Replace **<password>** with the password for the **fhsdjfh** user. Replace **<dbname>** with the name of the database that connections will use by default. Ensure any option params are [URL encoded](#).

Figure 22: Connect to the cluster

7. Click on collections to create the database.

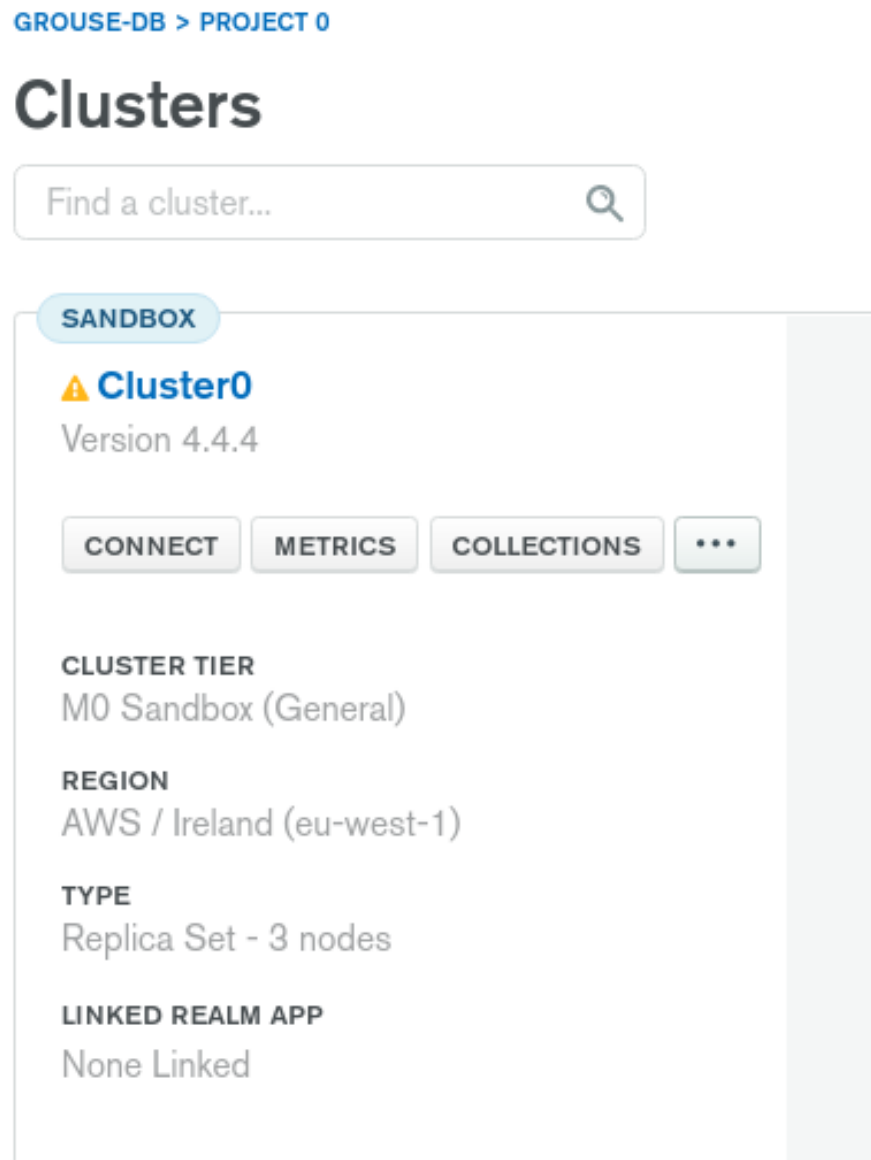


Figure 23: Create collections

8. Click on *Add My Own Data*.

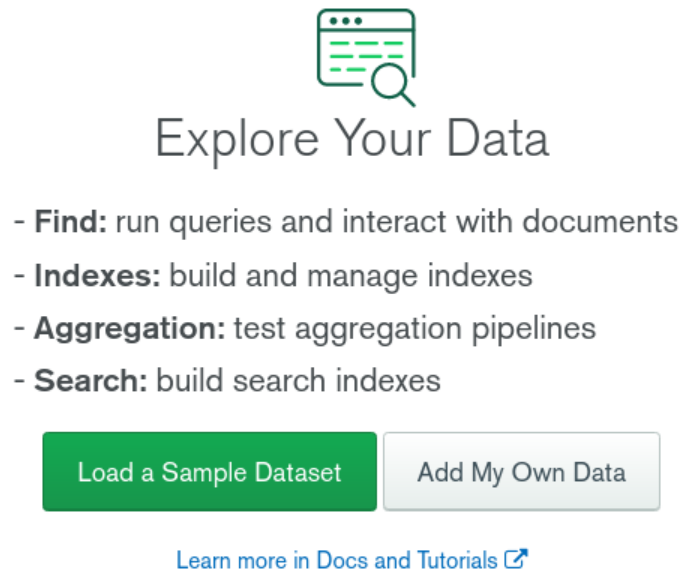
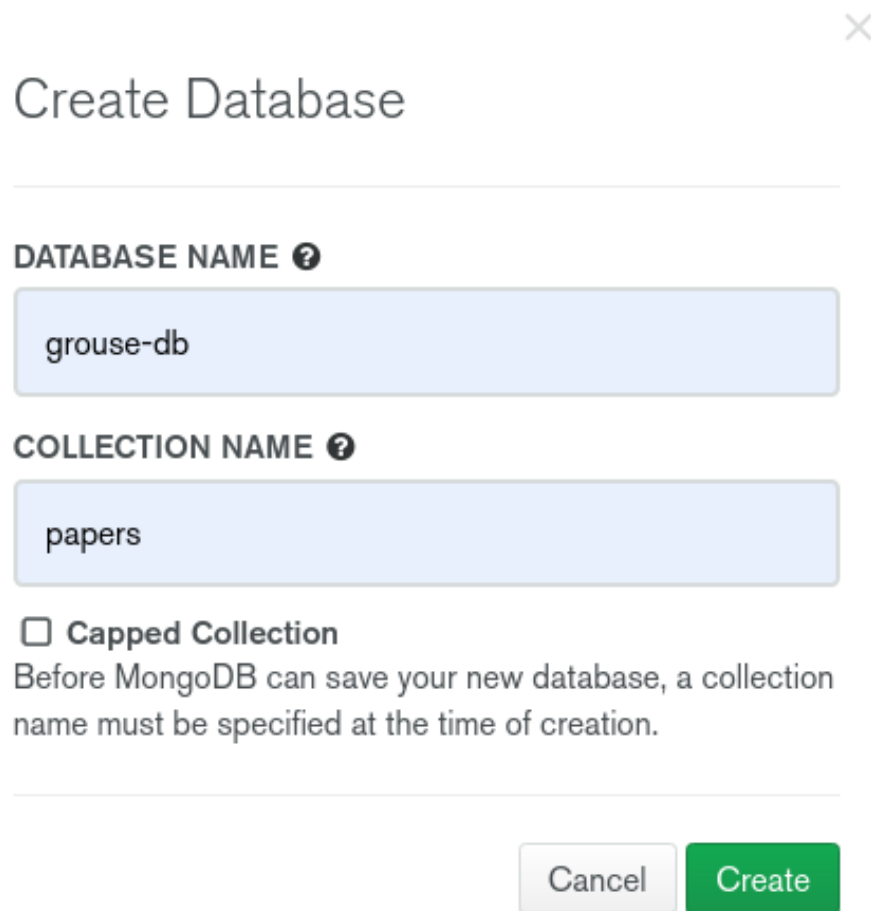


Figure 24: Add own data

9. Enter the database name as **grouse-db** and create collections as mentioned in section ??.



×

Create Database

DATABASE NAME ?

grouse-db

COLLECTION NAME ?

papers

☐ **Capped Collection**
Before MongoDB can save your new database, a collection name must be specified at the time of creation.

Cancel Create

Figure 25: Enter database name and collection name

7.1.2.1 grouse-db: Collections

First, we go through some of the key MongoDB concepts:

1. *Document*: A *document* is a unit of data in MongoDB and is akin to a row of a table in an RDBMS (45).
2. *Collection*: A *collection* is a grouping of documents. This can be thought of as a table in an RDBMS (46).

3. *Query*: Using a *query*, the selection criteria can be specified.
4. *Projection*: The *projection* specifies the fields of to return for all documents in a collection.
5. *Cursor*: A *cursor* is essentially a pointer to result set returned by some *query*.
6. *BSON*: BSON is a combination of the terms *binary* and *JSON*. This is a serialization format which is used to store documents (47).
7. *ObjectId*: A BSON type which uniquely identifies a document within a collection. It is composed of the timestamp, a random value and a counter (48).
8. *Operator*: An operator is a mongo keyword which starts with the \$ symbol. It helps in performing updates and query transformation operations. An example of this is \$lt which denotes the "less than" operator (49).
9. *Cluster*: A *cluster* is essentially a network of nodes on which a MongoDB deployment is done. A deployment primarily consists of sharded collections across nodes (50).

Next, we define some of the pertinent MongoDB collection methods and operators used in `grouse-server`.

Collection methods used:

1. `find(query, projection)`: This method returns a pointer to a collection of documents (51).
2. `findOne(query, projection)`: This method returns a single document which satisfies the criteria specified by the *query* (52).
3. `insertOne`: This method inserts a document into a collection (53).

4. `deleteOne`: Using this method, a document is removed from the collection (54).
5. `updateOne`: Using this method, a document is updated the collection (55).
6. `distinct`: Fetches all distinct values which a field in a collection takes and returns the results in an array (56).

Operators used:

1. `$regex`: All documents whose field's value(s) match a regular expression can be fetched using this operator.
2. `$in`: This operator matches a field's value present within an array of values.
3. `$set`: This operator is used to add new fields to a document in a collection.

We list down and define the following collections along with an example for each:

1. `categories`: All defined *categories* are stored in this collection. Here, the `category` key refers to the category string. `paper_references` is a list of `_id` which uniquely identifies a `paper` document. `paper_titles` is a list of all *paper titles* and `paper_terms` is a list of all *terms* categorized by the aforementioned category.

```

1 {
2   "_id":{"$oid":"6025e32c974a4024c245452d"},
3   "category":"Modelling",
4   "paper_references":["600cdb15d4677ead6f29ae13"],
5   "paper_titles":["An extensive dataset of UML models in GitHub"],
6   "paper_terms":{}
7 }
```

2. **ground-truth-terms**: This collection represents a *set* of only those terms which are to be included in the conceptual graph and operations pertaining to it. Here, the **term** key refers to the *term* string added to the ground truth.

```

1 {
2   "_id":{"$_oid":"60204872974a4024c245139f"},
3   "term":"a-priori"
4 }
```

3. **paper-category**: This collection stores all categories for a paper. Here, the **paper_title** key refers to the title of some paper. **category_references** is a list of **_id** which uniquely identifies a **category** document. **categories_annotated_with** is a list of all *categories* which categorize the aforementioned paper.

```

1 {
2   "_id":{"$_oid":"6025e342974a4024c245452e"},
3   "paper_title":"An extensive dataset of UML models in GitHub",
4   "category_references":["6025e32c974a4024c245452d"],
5   "categories_annotated_with":["Modelling"]
6 }
```

4. **papers**: This collection represents all *papers* in our corpus. **paper_title** is the title of a paper, **raw_text** is the *raw textual content* of the paper. **paper_terms** is a *dictionary* with key as the *term* and whose value is a *dictionary* with the **sentence_indices** key. **sentence_indices** is a list of integers which correspond to the *sentence nos.* in which the *term* appears.

```

1 {
2   "_id":{"$_id":"600cddb6d4677ead6f29aff6"},
3   "paper_title":"The Open-Closed Principle of Modern Machine Learning
4   Frameworks",
5   "raw_text":"The Open-Closed Principle of Modern Machine Learning
6   Frameworks\nThe Open-Closed Principle of Modern Machine Learning\n
7   Frameworks\nHoussem Ben Braiek\nSWAT Lab.",
8   "paper_terms":
9   {
10    "rising":
11    {
12    "sentence_indices": [16,170,176,325]

```

5. **relations:** This collection defines possible relation strings that can be used to relate two terms. Each document uses the **relation** key which corresponds to a *relation* string and the **terms_annotated** key stores a list of all terms which are related using this *relation*.

```

1 {
2   "_id":{"$_id":"602589ac974a4024c2454528"},
3   "relation":"create",
4   "terms_annotated":["apache","author"]
5 }

```


6. **term-relations**: This collection stores all possible *relations* for a *term*. The **term** key specifies the *term* string and the **relation_references** stores an array of dictionaries where each element has the **relation_reference** key which is the `_id` of the relation, **relation** which represents the *relation* string, **paper_title_primary** and **paper_title_secondary** which is the title of the paper to which the *outer term* and term defined in the *relation reference element* belong to respectively. The **inference_strategy** stores the context in which the relation is added.

```

1 {
2   "_id":{"$_id":"602589ac974a4024c245452a"},
3   "term":"author",
4   "relation_references":
5   [
6     {
7       "relation_reference":"602589ac974a4024c2454528",
8       "relation":"create",
9       "term":"apache",
10      "paper_title_primary":"How Distributed Version Control Systems
11      Impact Open Source Software Projects",
12      "paper_title_secondary":"How Distributed Version Control
13      Systems Impact Open Source Software Projects",
14      "inference_strategy":"GROUND_TRUTH"
15    }
16  ]
17 }

```

7.1.3 grouse-server: The backend

In this section, we describe what the pertinent files are, where they are located in the project directory and what they achieve.

7.1.3.1 application.py

This file defines the REST service endpoints associated with the application.

1. GET `/categories`: Fetches all categories.
2. POST `/category`: Adds a category.
3. POST `/paper/category`: Adds a category for a paper.
4. DELETE `/paper/category`: Deletes a category of the paper.
5. POST `/paper/terms/category`: Adds a category for a term in the paper.
6. DELETE `/paper/terms/category`: Deletes a category for a term in the paper.
7. GET `/relations`: Fetches all relations.
8. POST `/relation`: Adds a relation.
9. DELETE `/relation`: Deletes a relation.
10. POST `/term/relation`: Adds the specified relation for a term.
11. DELETE `/term/relation`: Deletes the specified relation for a term.
12. GET `/term/relation/term={term}`: Fetches all relations for the specified term.

7.1.3.2 grouse/schema.py

This file consists of schema which are Python classes corresponding to the resources used by the application. These schema include:

1. **CategorySchema**: This schema stores category information.
2. **PaperCategorySchema**: This schema stores all the categories assigned to a paper.
3. **RelationsSchema**: In an instance of this schema, the relation and all terms which are assigned this relation are stored.
4. **RelationReferencesSchema**: Such a schema depicts a relation between two terms. It consists of properties such as `term`, `relation_reference` which is the unique identifier for the relation, `relation` which is the string representation of it, `paper_title_secondary` which corresponds to the title of the paper which `term` belongs to, `paper_title_primary` which is the title of the paper which connects some term to `term` and the `inference_strategy` which is a string defining the strategy used to infer `relation`.
5. **TermRelationsSchema**: This schema defines properties such as `term` and a list of `RelationReferencesSchema`.

7.1.3.3 db/mongo_client.py

This file consists of the `MongoRepository` class which defines repository specific functions and it is here that the developer initializes arguments such as the connection URL, the database name and the password. A connection URL is a string which establishes a connection between your application and a MongoDB instance.

Some algorithms for pertinent functions such as adding a adding category for paper and a term, merging categories and adding a relation for a term.

7.1.3.4 sentence_parsing/utilities.py

The functions defined in this file are responsible for extracting terms and sentences from each paper and also the sentences in which a term appears.

7.1.4 Using the application

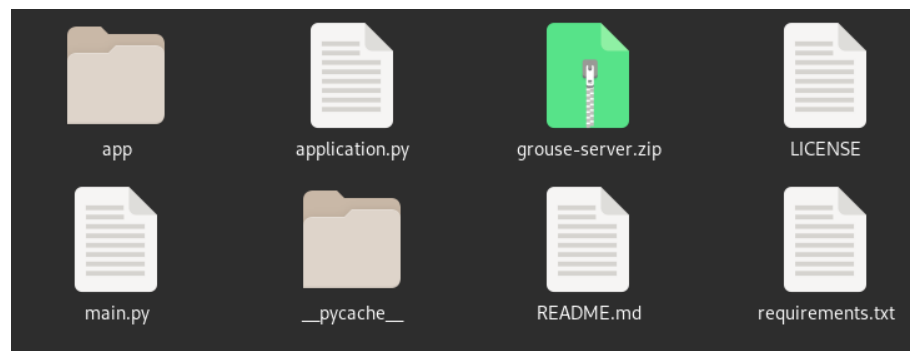
In order to use this application, the code for the user interface is to be cloned from the following GitHub repository. The instructions to spawn an instance of this UI are specified in the README in the repository. As of now this instance can only be used locally so it is to be ensured the *Node.js* runtime is set up locally.

The **grouse-server** can either be deployed on an *AWS EC2* instance or can be run locally. For our purposes, we made use of *AWS Elastic Beanstalk* which is a service provided by Amazon which automates the process of deployment and provides additional configuration options in terms of security groups to control who can access the instance, load balancing options when the application is to be deployed on multiple EC2 instances, type of EC2 instance etc.

7.1.4.1 Deploying grouse-server

The following steps outline how **grouse-server** can be deployed on an EC2 instance:

1. Clone the git repository from the following URL : <https://github.com/jeet1995/grouse-server>
2. Zip the contents of parent folder.

Figure 26: Parent directory of `grouse-server`

3. Open up the AWS Beanstalk console and click on the *Environments* option on the left followed by the *Create Environment* button as shown below:

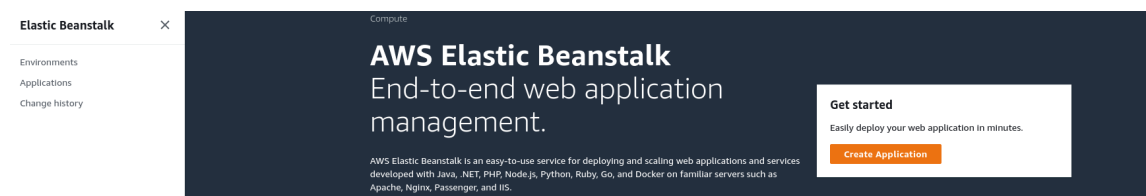


Figure 27: Create environment

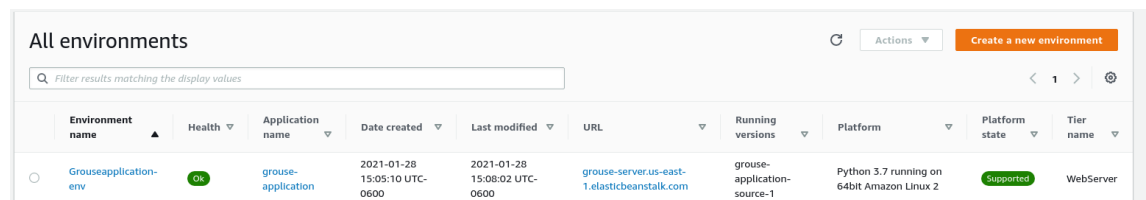


Figure 28: Create application

4. Select the environment tier as *web-server environment* as our application will run on a web-server. After this, fill in the application name as *grouse-application*

Select environment tier

AWS Elastic Beanstalk has two types of environment tiers to support different types of web applications. Web servers are standard applications that listen for and then process HTTP requests, typically over port 80. Workers are specialized applications that have a background processing task that listens for messages on an Amazon SQS queue. Worker applications post those messages to your application by using HTTP.

☒ **Web server environment**
Run a website, web application, or web API that serves HTTP requests.
[Learn more](#)

☐ **Worker environment**
Run a worker application that processes long-running workloads on demand or performs tasks on a schedule.
[Learn more](#)

Cancel Select

Figure 29: Select environment tier

Application information

Application name

Up to 100 Unicode characters, not including forward slash (/).

► Application tags (optional)

Environment information


Choose the name, subdomain, and description for your environment. These cannot be changed later.

Environment name

Domain

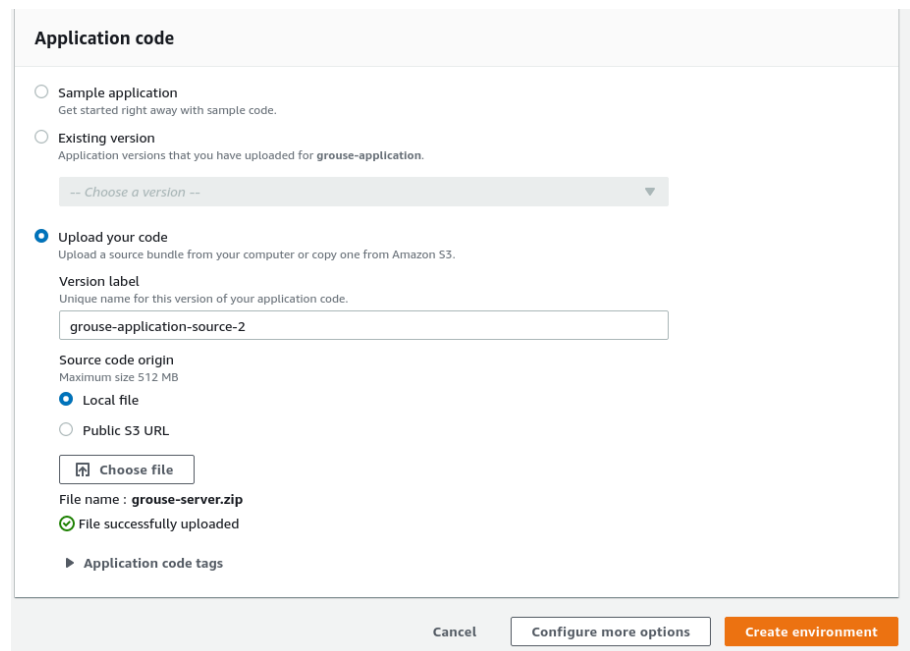
.us-east-1.elasticbeanstalk.

Check availability

 grouse-server.us-east-1.elasticbeanstalk.com **is unavailable.**

Description

- Figure 30: Filling in the application and environment name along with the domain
5. Upload the application code (grouse-server.zip) and click on *Configure More Options*.



Application code

☐ **Sample application**
Get started right away with sample code.

☐ **Existing version**
Application versions that you have uploaded for grouse-application.

-- Choose a version --

☒ **Upload your code**
Upload a source bundle from your computer or copy one from Amazon S3.

Version label
Unique name for this version of your application code.

grouse-application-source-2

Source code origin
Maximum size 512 MB

☒ **Local file**

☐ **Public S3 URL**

Choose file

File name : grouse-server.zip

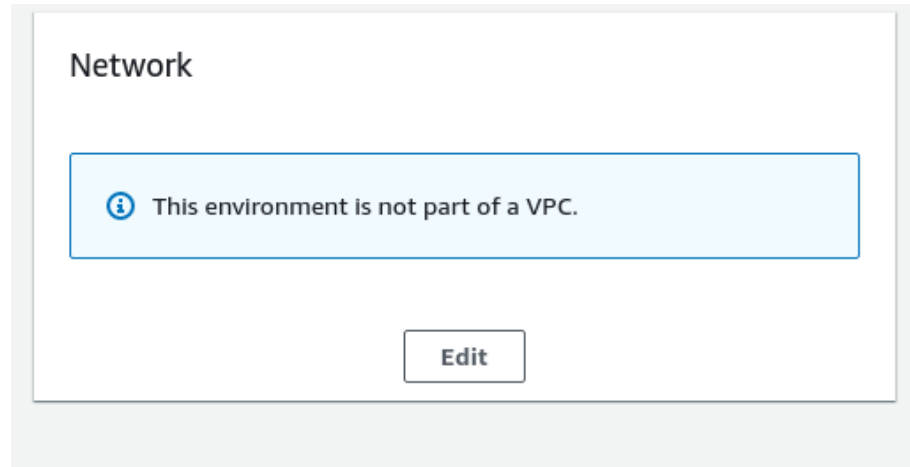
File successfully uploaded

Application code tags

Cancel Configure more options Create environment

Figure 31: Uploading application code

- Go to the *Network* card and select the default VPC and assign a public IP address to it.



Network

i This environment is not part of a VPC.

Edit

Figure 32: Edit Network card

Virtual private cloud (VPC)

VPC
Launch your environment in a custom VPC instead of the default VPC. You can create a VPC and subnets in the VPC management console. [Learn more](#)

vpc-a7058ac8 (10.0.0.0/16) | prestovpc

Your account doesn't have a default VPC for this region. [Create default VPC](#)

Instance settings

Choose a subnet in each AZ for the instances that run your application. To avoid exposing your instances to the Internet, run your instances in private subnets and load balancer in public subnets. To run your load balancer and instances in the same public subnets, assign public IP addresses to the instances.

☒ **Public IP address**
Assign a public IP address to the Amazon EC2 instances in your environment.

Instance subnets

<input checked="" type="checkbox"/>	Availability Zone	Subnet	CIDR	Name
<input checked="" type="checkbox"/>	us-east-1a	subnet-bd058ad2	10.0.0.0/24	

Cancel

Save

Figure 33: Configuring the VPC and the instance subnet

It is important to ensure that the domain is available and the `baseURL` string present in the service classes of `grouse-ui` should be initialized with the same domain.

CHAPTER 8

INFERENCE TECHNIQUES

The objective of this thesis is to construct a conceptual graph for the corpus under interest. We identify categories, relations, papers and terms these papers contain as the elements of our graph. We outline some of the steps we plan on taking to infer these elements using various topic modelling techniques and other approaches pertaining to research question extraction, PoS tagging, term expansion and citation network extraction.

8.1 Building our collections

We first require the user of our application to persist all documents along with terms into the papers collection. This implies extracting its titles, words, sentences and the sentences wherein these words appear.

8.2 Building the ground truth conceptual graph

In this step, a domain expert is required to categorize documents, terms and specify relations between terms across papers. A prerequisite step that can be carried out is to select words of greater focus so as to simplify conceptual graph creation.

8.3 Concept graph creation through term expansion

In this technique, the constituent terms of every relation can be iterated through and can be augmented with other terms which are semantically synonymous. An example is, say there exists a term relation, $TR = \{ 'term1': 'communities', 'relation': 'possesses', 'term2': 'knowl-$

edge’}. An inferred term relation could be $\text{TR}_{\text{inferred}} = \{ \text{'term1': 'community', 'relation': 'possesses', 'term2': 'knowledge'} \}$.

8.4 Concept graph creation using research questions

This approach will be used when nouns implicating each other can be extracted using part-of-speech tagging of research questions. An example could be "How does drinking coffee affect developer productivity?". Here, part-of-speech (*PoS*) tagging would extract nouns set $S = \{ \text{"coffee", "developer", "productivity"} \}$ and a relation could be generated between a pair of words belonging to this set.

8.5 Concept graph creation through Relational Topic Modelling

When we talk about linking papers, we first need to categorize similar papers or in other words, we need to create a document network. The labels could be either created by adding a new category which can be further used to annotate a paper. In our network, a node v corresponds to a paper and an undirected edge e corresponds to links between them.

Following this, we can make use of the *predictive label rank* (PLR) as a measure to determine how strongly linked two papers are (57). Observing the topic assignment of a document that has been drawn out, the probability with which it will link to another document will be computed as mentioned in section 4.2.3.4. Then the PLR is computed as follows (57):

$$\text{PLR}(d) = \frac{1}{D} \sum_{(d,d') \in E} r_{d'} \quad (8.1)$$

This process can be repeated with a trained RTM with 10 randomly chosen subsets of data. Each time, 9 of these subsets would be used for training and the remaining would be used for testing (57). The average PLR across all repetitions can be calculated and it can be determined what the best cluster for these papers are (57). The domain expert can now focus on these clusters to create new relations between terms belonging to a pair of papers in this cluster (57).

8.6 Concept graph creation through Latent Semantic Analysis

The approach here is to first cluster terms in a paper into topics. Then, we build a query with terms belonging to some topic. This in turn will give us documents which are closely characterized by the query terms. Hence, this will form as a precursor to group related documents and to identify terms between these documents to be related.

8.7 Concept graph creation through Expected Entropy Loss

Here, we again start of by first having a domain expert to run through documents and to define possible categories which could be used to classify the entire corpus. Then the next step is to categorize a subset of papers. What *EEL* helps in realizing is to what extent these terms pertaining to the categorized papers can be categorized with a certain category C. The domain expert can then pick out terms from a given category and add relations between them.

In case the corpus of papers is too large for manual annotation, supervised learning approaches such as *Naive Bayes*, *Decision Trees* and *Support Vector Machines* can be used to categorize papers in the testing set (33). These approaches will include a feature vector set which realizes all terms across the corpus of papers and how often they occur.

CHAPTER 9

CONCLUSION

In this thesis, we achieve the objective of creating a conceptual graph. This is done by eliciting from a domain expert categories of each paper in the corpus and the terms these papers may contain. Relations which are represented in the form of verb phrases are used to relate two terms within the same paper or across different papers. These annotations are elicited using an Angular based user interface and are persisted to a MongoDB database through a flask based server written in the Python programming language.

The benefits of creating such a conceptual graph are drawing inferences across a corpus of research papers which can help formulate research questions aiding meta-research publications. Our ground truth conceptual graph can be used as a training dataset to draw more inferences with a high degree of automation with the help of topic modellers and classification algorithms.

CITED LITERATURE

1. Sowa, J. F.: Conceptual graphs. Foundations of Artificial Intelligence, 3:213–237, 2008.
2. Sena, D., Coelho, R., Kulesza, U., and Bonifácio, R.: Understanding the exception handling strategies of java libraries: An empirical study. In Proceedings of the 13th International Conference on Mining Software Repositories, MSR '16, page 212–222, New York, NY, USA, 2016. Association for Computing Machinery.
3. Avery, D., Dam, H. K., Savarimuthu, B. T. R., and Ghose, A.: Externalization of software behavior by the mining of norms. In Proceedings of the 13th International Conference on Mining Software Repositories, MSR '16, page 223–234, New York, NY, USA, 2016. Association for Computing Machinery.
4. Ortu, M., Adams, B., Destefanis, G., Tourani, P., Marchesi, M., and Tonelli, R.: Are bullies more productive? empirical study of affectiveness vs. issue fixing time. 05 2015.
5. Campbell, J. C., Santos, E. A., and Hindle, A.: The unreasonable effectiveness of traditional information retrieval in crash report deduplication. In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pages 269–280, 2016.
6. Blaz, C. C. A. and Becker, K.: Sentiment analysis in tickets for it support. In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pages 235–246, 2016.
7. Storey, V. C.: Understanding semantic relationships. The VLDB Journal, 2:455–488, 10 1993.
8. Hensman, S.: Construction of conceptual graph representation of texts. In Proceedings of the Student Research Workshop at HLT-NAACL 2004, HLT-SRWS '04, page 49–54, USA, 2004. Association for Computational Linguistics.
9. Hasegawa, R., Kitamura, M., Kaiya, H., and Saeki, M.: Extracting conceptual graphs from japanese documents for software requirements modeling. In Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling - Volume 96, APCCM '09, page 87–96, AUS, 2009. Australian Computer Society, Inc.

10. Ma'ayan, D., Ni, W., Ye, K., Kulkarni, C., and Sunshine, J.: How domain experts create conceptual diagrams and implications for tool design. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, CHI '20, page 1–14, New York, NY, USA, 2020. Association for Computing Machinery.
11. Hines, T. R. and Unger, E. A.: A variation of conceptual graphs: An object-oriented approach. In Proceedings of the 1987 Fall Joint Computer Conference on Exploring Technology: Today and Tomorrow, ACM '87, page 57–68, Washington, DC, USA, 1987. IEEE Computer Society Press.
12. Nečaský, M.: Reverse engineering of xml schemas to conceptual diagrams. In Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling - Volume 96, APCCM '09, page 117–128, AUS, 2009. Australian Computer Society, Inc.
13. Benda, S., Klímek, J., and Nečaský, M.: Using schematron as schema language in conceptual modeling for xml. In Proceedings of the Ninth Asia-Pacific Conference on Conceptual Modelling - Volume 143, APCCM '13, page 31–40, AUS, 2013. Australian Computer Society, Inc.
14. Li, X. and Parsons, J.: Ontological semantics for the use of uml in conceptual modeling. In Tutorials, Posters, Panels and Industrial Contributions at the 26th International Conference on Conceptual Modeling - Volume 83, ER '07, page 179–184, AUS, 2007. Australian Computer Society, Inc.
15. Bouchakwa, M., Ayadi, Y., and Amous, I.: Modeling the semantic content of the socio-tagged images based on the extended conceptual graphs formalism. In Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media, MoMM '16, page 35–39, New York, NY, USA, 2016. Association for Computing Machinery.
16. Demeure, A. and Calvary, G.: Plasticity of user interfaces: towards an evolution model based on conceptual graphs. In Proceedings of the 15th Conference on l'Interaction Homme-Machine, pages 80–87, 2003.
17. Chernyak, E.: An approach to the problem of annotation of research publications. In Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, WSDM '15, page 429–434, New York, NY, USA, 2015. Association for Computing Machinery.

18. Jo, Y., Kim, E.-G., and Shin, Y.: Graphical keyword service for research papers with text-mining method. In Proceedings of the International Conference on Compute and Data Analysis, ICCDA '17, page 185–190, New York, NY, USA, 2017. Association for Computing Machinery.
19. Ensan, F. and Bagheri, E.: Document retrieval model through semantic linking. In Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM '17, page 181–190, New York, NY, USA, 2017. Association for Computing Machinery.
20. Kochedykov, D., Apishev, M., Golitsyn, L., and Vorontsov, K.: Fast and modular regularized topic modelling. In 2017 21st Conference of Open Innovations Association (FRUCT), pages 182–193, 2017.
21. O'Neill, J., Robin, C., O'Brien, L., and Buitelaar, P.: An analysis of topic modelling for legislative texts. CEUR Workshop Proceedings, 2016.
22. Lim, K. H., Karunasekera, S., and Harwood, A.: Clustop: A clustering-based topic modelling algorithm for twitter using word networks. In 2017 IEEE International Conference on Big Data (Big Data), pages 2009–2018, 2017.
23. Hamedani, M. R., Lee, S.-C., and Kim, S.-W.: On combining text-based and link-based similarity measures for scientific papers. In Proceedings of the 2013 Research in Adaptive and Convergent Systems, RACS '13, page 111–115, New York, NY, USA, 2013. Association for Computing Machinery.
24. Mihalcea, R. and Csomai, A.: Wikify! linking documents to encyclopedic knowledge. In Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management, CIKM '07, page 233–242, New York, NY, USA, 2007. Association for Computing Machinery.
25. Erol, B. and Hull, J. J.: Linking presentation documents using image analysis. In The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003, volume 1, pages 97–101 Vol.1, 2003.
26. Chakaravarthy, V. T., Gupta, H., Roy, P., and Mohania, M.: Efficiently linking text documents with relevant structured information. In Proceedings of the 32nd international conference on Very large data bases, pages 667–678, 2006.

27. Shaffer, R. and Mayhew, S.: Legal linking: citation resolution and suggestion in constitutional law. In Proceedings of the Natural Legal Language Processing Workshop 2019, pages 39–44, 2019.
28. Tokenization. <https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html>.
29. Toutanova, K., Klein, D., Manning, C. D., and Singer, Y.: Feature-rich part-of-speech tagging with a cyclic dependency network. In Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, pages 252–259, 2003.
30. Term expansion. https://docs.oracle.com/cd/E26180_01/Search.94/ATGSearchQueryRef/html/s0203termexpansion01.html#:~:text=Term%20expansion%20refers%20to%20the,or%20some%20other%20similar%20data.
31. Thomo, A.: Latent semantic analysis. pages 1–7.
32. Chang, J. and Blei, D.: Relational topic models for document networks. In Artificial intelligence and statistics, pages 81–88. PMLR, 2009.
33. McMillan, C., Linares-Vásquez, M., Poshyvanyk, D., and Grechanik, M.: Categorizing software applications for maintenance. In 2011 27th IEEE International Conference on Software Maintenance (ICSM), pages 343–352, 2011.
34. Natural language toolkit. <https://www.nltk.org/>.
35. Fellbaum, C.: Wordnet. In Theory and applications of ontology: computer applications, pages 231–243. Springer, 2010.
36. Apache pdfbox. <https://pdfbox.apache.org/>.
37. Welcome to flask - flask documentation. <https://flask.palletsprojects.com/en/1.1.x/>.
38. marshmallow: simplified object creation. <https://marshmallow.readthedocs.io/en/stable/>.
39. Getting started - scikit-learn. https://scikit-learn.org/stable/getting_started.html.

40. Allenai: science-parse. <https://github.com/allenai/science-parse>.
41. Pymongo documentation. <https://pymongo.readthedocs.io/en/stable/>.
42. Mining software repositories. <http://www.msrrconf.org/>.
43. Angular component. <https://angular.io/api/core/Component#description>.
44. Angular cdk scrolling component. <https://material.angular.io/cdk/scrolling/overview>.
45. Mongodb document. <https://docs.mongodb.com/manual/core/document/>.
46. Mongodb databases and collections. <https://docs.mongodb.com/manual/core/databases-and-collections/>.
47. Mongodb json and bson. <https://www.mongodb.com/json-and-bson>.
48. Mongodb objectid. <https://docs.mongodb.com/manual/reference/method/ObjectId/>.
49. Mongodb operator. <https://docs.mongodb.com/manual/reference/operator/>.
50. Mongodb cluster. <https://docs.mongodb.com/manual/core/sharded-cluster-components/>.
51. Mongodb collection method: find. <https://docs.mongodb.com/manual/reference/method/db.collection.find/>.
52. Mongodb collection method: findone. <https://docs.mongodb.com/manual/reference/method/db.collection.findOne/>.
53. Mongodb collection method: insertone. <https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/>.
54. Mongodb collection method: deleteone. <https://docs.mongodb.com/manual/reference/method/db.collection.deleteOne/>.
55. Mongodb collection method: updateone. <https://docs.mongodb.com/manual/reference/method/db.collection.updateOne/>.

56. Mongodb collection method: distinct. <https://docs.mongodb.com/manual/reference/method/db.collection.distinct/>.
57. Lu, Y. and Hao, S.: Relational topic model for congressional bills corpus. pages 1–12, 2016.

VITA

NAME	Abhijeet Mohanty
EDUCATION	B.Tech in Computer Engineering, National Institute of Technology, Karnataka, 2017
EXPERIENCE	Graduate Research Assistant, University of Illinois at Chicago, IL, Jan 2020 – May 2020 Specialist Software Engineer, Societe Generale Global Solutions Center, India, Aug 2017 – Jul 2019
TEACHING	Graduate Teaching Assistant (CS 441 - Engineering Distributed Objects for Cloud Computing, Fall 2020)